Infrastructure for mobile agents

Yolande Berbers, Bart De Decker, Wouter Joosen
Department of Computer Science, KULeuven
email: Yolande.Berbers@cs.kuleuven.ac.be
snail mail: Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

State of the art distributed application technology is not suited to tackle the challenges offered by the "Information Superhighway". A new communication model based on mobile agents has been proposed and is still intensely studied. Research on the software infrastructure required to host agents on the move is urgently needed.
The paper presents a requirements analysis for the infrastructure needs of agents in a distributed applications environment such as DCE or CORBA. It introduces an outline of a comprehensive proposal which includes requirements for autonomy, communication, mobility and security.

1. High-bandwidth office environments versus worldwide network systems

The explosive growth of the Internet drastically changes the possibilities offered to computer users. The new **worldwide network systems** differ in several manifest aspects from traditional high-bandwidth office distributed systems:

- The number of available electronic services from businesses and governments and the volume of available information are several orders of magnitude larger than in traditional distributed systems.
- Connectivity between nodes is highly variable both in performance and in reliability; the variation in bandwidth and latency is large; connectivity may be intermittent.
- Mobility plays an important role: users may plug in their computers on the net at different places at different times; they may be working off-line a large part of the time as communication may be expensive.
- Where in traditional distributed systems nodes are primarily servers or workstations, mobile nodes usually have less resources (e.g. laptop computers with more limited storage and processing capabilities than workstations).

In the traditional world of distributed systems, the client/server model using Remote Procedure Calls (RPC's) has been accepted as the best model to cope with distribution, and large applications have been built using this technology. Systems such as DCE [1] constitute a standard that allows complex applications to be built across heterogeneous platforms.
The conventional client/server model has been adapted to the object-oriented technology, giving rise to systems such as CORBA [2] where objects are used both for clients and servers, and where RPC's are replaced by object invocations. Although both the traditional client/server model as the distributed object-oriented model perform well in a controlled, high-bandwidth office environment, they are not adapted to face the challenges of worldwide network systems.

In the new worldwide network environment, the way applications are designed needs to be rethought in order to allow applications to keep running when the user has a slow, unreliable connection, when the user changes location, and when the user is disconnected for some time. Clients should rely on servers when possible but function autonomously if needed. Especially the *middleware* that is used in building applications needs to be adapted to a worldwide network system. More specifically, a *different communication model* must be used that enhances the basic client/server model.

The rapid evolution of the Internet often leads to operational solutions with a relatively ad hoc approach/architecture. Sometimes, excellent and widely appreciated standards emerge from such an evolution (take DNS as an example), but this is certainly not the general case (take sendmail as an example). This kind of research focuses on prototyping with the advantage that working systems emerge fast, which is very useful for experiments.

Complementary research attempts to find innovative and conceptually new solutions. Try-outs typically occur on a smaller scale; the prototypes that are being built here are not of general use, and usually run in only one or a few environments. However, the solutions that prove to be worthwhile often find their way in future standards.

This paper should be seen as a contribution to yet another (a third) type of research: we try to address standards that may become widely accepted by a broad community. Good examples are work around DCE and CORBA; such work progresses slowly as it is based on consensus between a vast amount of people; however, it is well thought-out and proposes solutions that can be used in many real world environments.

This paper introduces agents as a communication paradigm that is better suited than traditional paradigms (e.g. RPC) in a worldwide network environment. The text offers a requirements analysis of the infrastructure needs for agents in distributed applications environments, without imposing assumptions about the systems that would be equipped with such an infrastructure. As such, it can serve as a guideline when establishing enhanced standards for distributed systems.

2. Mobile agents: a communication model for worldwide network systems

A new model for gluing together pieces of worldwide applications has been proposed under the name of "mobile agents" [3, 4, 5]. This communication paradigm has been popularized in recent years by researchers and developers interested in various applications ranging from smart document search, over shopping in the Internet, to intelligent network services, as well as in languages that support these applications.

The mobile agent paradigm should be seen in contrast to the traditional RPC model. In the RPC model, a client communicates with a non-local server. In the mobile agent paradigm, a typical agent travels to a server's host, interacts with this server, and returns to the client. The major difference between an RPC and an agent is that an RPC is passive, it is a low level mechanism that supports client/server interaction but that is subordinate to the client and the server. An agent on the contrary is a first class object. It is active, it has an own identity. It acts on the same abstraction level as the client and the server. In fact, clients, servers and their interactions are unified in the single concept of an agent.

A major advantage here lies in the client's ability to develop an agent request while disconnected, launch the agent during a brief connection session, and then immediately disconnect. The response, if any, is collected during a subsequent connection session. More complex mobile agents can access several

services on different nodes during one single journey, and can collaborate or negotiate with other agents, before returning to the clients who sent them. In this respect, mobile agents are a form of dynamic distributed computation enabling a more efficient division of labor among computing elements.

Some complex agents are intelligent. For example they can perform filtering at the server side, and return to the client only relevant information. In this way the information transmitted over the network is minimized. Filtering of information is just one case of additional functionalities that an agent can offer. Intelligent mobile agents employ some knowledge or representation of the user's goals or desires to carry out their task. Their intelligence is the degree of reasoning and learned behavior they exhibit.

The following characteristics of mobile agents determine the infrastructure necessary to host them while they are traveling. Agents are

- autonomous: agents are able to take initiatives, exercise control over their own actions and can be long-term running processes.
- communicative: agents are able to engage in complex communication with services or with other agents.
- mobile: agents are able to transport themselves from one machine to another and across different system architectures and platforms.
- self-starting: unlike standard programs which are directly invoked by the user, an agent can sense changes to its environment and decide when to act.
- goal-oriented: an agent accepts high-level requests indicating what a client (or the human behind the client program) wants and is responsible for deciding how and where to satisfy the request.

3. Infrastructure for mobile agents

Agents need a solid infrastructure in the form of local environments that welcome them and help them to perform their tasks. We call such environments **agencies**. We expect that in the future, agencies will be part of the basic infrastructure provided by each node connected to the Internet.
As heterogeneity and inter-operability are inherent to worldwide network systems, openness and multi-standard compliance must be key properties of any infrastructure for agents.
The basic tasks of agencies can be summarized as follows:

1. The most basic services that agencies provide are facilities for agents to execute, i.e. to allocate resources if allowed and to enable autonomous behavior afterwards. These basic services are offered through the runtime system.
2. Agents must communicate with other agents and with servers, both local and remote. Agencies must support these needs.
3. Agencies provide facilities for agent traveling, and should support this in a heterogeneous environment. Special attention must be paid to state information.
4. Security is undoubtedly an important aspect. Both agencies and agents must be able to protect themselves and their interactions against eavesdropping, tampering and other kinds of misbehavior. Means for mutual authentication, authorization, privacy, integrity control and digital signatures must be available, so that any level of reassurance can be obtained, access to resources can be protected and services such as electronic cash can be built.

In the next paragraphs, each of these basic tasks will be dealt with in more detail. We will use the analogy of a hotel to describe the way agencies work: hotels offer a variety of services for their guests, the agents in our analogy.

3.1 Execution facilities

Agents arriving at an agency must be admitted. This is the task of the **door keeper**. The door keeper can receive agents through various protocols, and unpacks them. A traveling agent consists of code, state and various attributes describing the identity of an agent, its needs, and other characteristics. Unpacking includes extracting a.o. the credentials from the attributes for security checking (see further). If the agent passes this security check, the door keeper hands the agent over to the **check in desk**.

The main task of the check in desk is to allocate resources. The check in desk examines the attributes to find out more about the needs of the guest, in order to assign an appropriate room. This includes first of all checking the language the agent is written in.
Two classes of solutions exist for programming languages for agents: dynamic compilation and interpretation. Both have their advantages and disadvantages.

- In the first solution, the code of an agent is transported in the form of source code, which is dynamically compiled at each location where the agent is moving to, and then locally executed. Dynamic compilation costs time, but has the advantage that the generated code can be optimized with respect to the environment. A further advantage of this scheme is that source code can be inspected by the security check, but a disadvantage is that many companies are not too happy with giving away their source code.
  In theory all languages can be used; in practice however, the compilation of some languages is not completely platform independent (e.g. C). Self and CORRELATE [6] are examples of languages suited for dynamic compilation.
- Much attention is currently paid to languages which are either interpreted directly or which are compiled to a portable intermediate code which is interpreted. Tcl [7] and Perl [8] are examples of languages that are directly interpreted, Java [9] is transported as intermediate byte code. For such languages, an appropriate interpreter must be available in each agency. Interpreting a program is of course less efficient than executing machine code; however the code does not have to be compiled by the agency.

For reasons of openness and multi-standard compliance a general agency in an Internet environment should not limit itself to one language, but must support a variety of languages suited to program agents, both languages suited for dynamic compilation and languages that are interpreted.

Further attributes the check in desk will look at are the memory needs of the agent, or limitations such as expiration date. Other resource needs, such as necessary services available at the agency, can also be stated in the attributes. However, it is more reasonable for an agent to check such needs before moving, as the best action the check in desk can perform when an absolutely necessary service is not available, is to reject the agent. In the attributes the check in desk will find information on which actions should be undertaken on error, e.g. reject or discard the agent, deliver an error notification to a specified address, or routing the agent to yet another agency. An absolutely necessary service which could be missing is e.g. an interpreter for a specific language. If such interpreter is not available, the agent can of course impossibly execute.

A last important task of the check in desk is registering the agent in the agency so that the agent can be known to other guests. We will talk about this more in detail in the section on communication.

In the previous paragraphs we have supposed that an agent has come from an other node. Of course an agent can also be created on the node itself. The same mechanisms apply to this case. The difference is

that the door keeper does not receive a traveling agent from over the network but receives all parts (code, state and attributes) from a local entity, which could be, but is not necessarily, an agent.

Run-time support must be available for operations such as moving, communication, and inspecting and changing an agents' attributes. For each language that is supported by an agency, a run-time package must be written that will be linked with dynamically compiled code or that is part of an interpreter.

3.2 Communication facilities

Part of the basic services offered by an agency are communication and meeting services. Agents must be able to communicate with services at the host system, with other agents in the agency and in other agencies, and also with its owner (which could be another agent). For reasons of openness, standard communication mechanisms such as RPC and object invocation must be supported. In addition to these, higher level interactions can be described as mobile agents that perform multicast-like interactions that are based on conditions, for instance by evaluating attributes of the potential callees.

To help agents locate other agents and services, an agency must provide an **information desk**. Agents are registered in this information desk by the check in desk. Besides having information on local agents, the information desk keeps track of services offered on the local host, and communicates with other information desks of other agencies to trace agents and services on other nodes. Naming services such as DCE's Directory Service or the broker facilities of CORBA can be used for this purpose. Such naming services could themselves be implemented as agents.

Information about the owner of the agent can be found in the attributes. This could be another agent (e.g. that has spawned various agents) or a person. In the latter case, an e-mail address can be provided and asynchronous communication with the owner via e-mail is possible. As the owner might not continuously be connected, this type of communication is very appropriate. Information about the owner of an agent is very important in shopping applications where owners have to receive a bill.

When an agent wants to communicate with a non-local service it has to take a policy decision: it can use low budget standard communication or it can create (spawn) an agent to realize the interaction. In both cases, the agent responsible for the interaction can decide to move to the host where the required service resides. It is not really a choice between just two possibilities (remote communication or moving to the remote host) because complex interactions can be recursively defined as agents. A range of possibilities with different performance characteristics may be supported amongst which must be chosen. Taking a good policy decision can be quite complex.
First of all, moving depends on the possibilities offered by the remote host (e.g. is the programming language of the agent supported at the remote host). Secondly the expected communication overhead must be analyzed: e.g. if the service requires lots of information to be exchanged between the agent and the server, then moving to the host of the server might be interesting. However, if only a few parameters are sent with the service request and not much more is expected back, then remote communication will be more effective. The communication overhead is of course function of the specific communication implementation chosen, and this should be taken into account.
It must be clear that not all agents will carry the intelligence to perform these complex policy choices. The agency can provide services that help agents in their decision making. A remote communication request could be sent to a location control agent who either sends the agent to the remote host or chooses the most appropriate remote communication.

3.3 Traveling facilities

Migration of processes and objects are mechanisms that have been studied extensively in distributed systems and that have been successfully implemented in various systems (e.g. for load balancing purposes). Experience from systems offering object migration such as Emerald [10] or XENOOPS [11, 12] can be used in this respect. We consider here both systems in which processes and objects are orthogonal (like in Emerald), and systems with concurrent objects, i.e. in which the concepts of objects and processes are unified.

Moving an object or process involves moving its code and its state (including data). In traditional migration, the most difficult part in the migration mechanism is moving state. This state must be perfectly preserved as the object or process should not notice that it is being moved and it should be able to transparently continue its execution on the new host. Part of the state is platform dependent (e.g. addresses on the stack, registers), which means that it is difficult to transfer it in a heterogeneous environment. Other parts can more easily be transported. In any case, the situation with traveling agents is different: whereas in traditional systems offering migration *the system* decides when which process is migrated and to where, in the agent world it is *the agent* itself that can take the decision to travel. This has major consequences because the agent may decide to move when the state is holding the minimal set of processor abstractions.

Moving is for an agent a major break in its activities. The agent will often start something new at the new location. In such a case, an agent who decides to move can easily clean up its state before leaving in order to avoid the transfer of platform dependent state. The state variables that are migrated will only hold the accumulated results and knowledge of the agent.

The decision to which node an agent wants to travel can be made in different ways: the most simple situation is that the agent starts up with a static list of hosts it wants to visit, and just visits the hosts of the list, one by one. However, the agent can be more intelligent and make dynamic choices. These choices can be based on available services which can be checked dynamically (see the communication facilities), and can also be based on the state of an agent (e.g. depending on the outcome of actions an agent has already undertaken or on the information that an agent has already accumulated). Complex decision making based on artificial intelligence is possible and is a research topic on its own that receives attention in distributed AI.

Once an agent has decided to travel to another agency, different parts of the agency must be informed and must take proper actions: the information desk will delete the agent from the local agent list, the door keeper will pack the agent (code, state, attributes) and send the agent over the network using some standard protocol, and the check in desk can reclaim the resources that were used.

3.4 Security aspects

Security is an important aspect of agent-based applications and the environment in which they execute. Agencies provide execution facilities for agents that possibly originated elsewhere. As explained before, the **door keeper** will first inspect arriving agents before they are passed to the check in desk. Such an inspection might involve several steps:

- agents owned by users or coming from locations that caused problems previously, can be rejected;
- agents that "look" suspicious (i.e. contain doubtful code) undergo the same treatment. (In some hotels, you are not allowed to enter without a proper suit and tie.)

The **check in desk** might further check the agent's identity, ask for credentials or a membership card, as some agencies are only open to a closed user-group. Also, the agent might verify the trustworthiness of

the agency, in order to avoid being lured by a beautiful facade that hides a criminal organization. To that end, agencies can join "agency chains" that guarantee a certain quality. In that case, a visiting agent may be reassured about the service, if the agency can prove to belong to such a chain. Moreover, an independent organization might issue "stars" to agencies, based on the quality of service they provide, ... During check-in the agent and agency may be involved in a negotiation about class of service, prize, etc.

Information and services provided by other agents, are valuable resources that need to be protected and possibly paid for when requested. During an interaction, both the requester and the provider must be able to verify the identity of the other party. The provider must be able to apply proper access control to its resources, and must be reassured about the solvency of the requester. On the other hand, the requester wants to avoid interactions with Trojan providers. Hence, proper *(mutual) authentication and authorization* means are essential. These means should be very general, and may involve other agents (e.g. a trusted third party agent, a credit-agent, ...), certificates that contain forwarded access rights, etc.

The conversation between agents must sometimes be protected against eavesdropping and/or tampering. This is certainly the case when confidential or valuable information is exchanged. Mechanisms must therefore be provided to protect the *privacy and/or integrity* of any (local or remote) conversation.

Finally, agents must also be able to *protect their internal state* against prying or tampering. This is important in applications that carry electronic money, or new software versions that must be installed remotely, etc. Complete assurance is not possible, however, since the agency itself may misbehave, and modify or misinterpret the agent-code. Fault tolerance mechanisms should be used to protect against these kind of attacks.

If you consider an agent as a computing entity that does something useful and has an external behavior (i.e. the way it presents itself to the environment), then most of the security issues can be put in the behavioral part.

4. A view on ongoing work

The requirements that have been enumerated in this paper address a general infrastructure for mobile agents on the Internet. We believe that these requirements should lead to extensions to standard systems, such as CORBA and DCE. To contribute to a research activity that expands an open standard, we have not assumed any specific paradigm or restriction that would reduce the scope of applicability. However, it is important to evaluate research projects in which wide applicability is not the major concern. Such projects can take the liberty to experiment with new concepts and ideas, and the most appealing results may become major ingredients in the standard systems of tomorrow.

For instance, we wish to refer to the CORRELATE project of our research group [6, 12, 13]. CORRELATE is a concurrent object-oriented language with a meta-level architecture. In CORRELATE, agents are defined as a natural extension of active objects. An active object model unifies the concepts of processes and passive objects. Active objects have the capability to synchronize the execution of incoming requests from other objects. In this paradigm, the migration of an agent becomes an operation on the agent. This has an important impact: even if the agent delegates the execution of a migration policy, it still has the autonomy to synchronize the actual execution of the mechanism with other (ongoing or pending) operations. This obviously simplifies the implementation of a mobility support layer (cfr. section 3.3) [12], and therefore, it also increases the potential for interoperability.

The fundamental issue emerging from this observation is the question whether or not a more standardized (and maybe narrowing) definition of process abstractions should become part of standard broker architectures.

We do not want to stress the single example of CORRELATE, we rather try to illustrate the way research projects should be evaluated in the framework of the requirements that have been grouped in this paper. Other relevant projects that are important in this perspective include work on systems for agents such as TACOMA [14] and of course work on languages such as Telescript [15], TcL [7, 16, 17], and Java [9], just to name a few.

5. Conclusion

This paper first introduces agents as a communication paradigm that is better suited than traditional paradigms (e.g. RPC) in a worldwide network environment. It then offers a requirements analysis of the infrastructure needs of agents in distributed applications environments such as DCE or CORBA. It offers an overview of the facilities needed to support the autonomous aspects of agents (execution facilities), the communicative and mobile aspects of agents, and also the security aspects. Specific security needs are a direct consequence of the fact that agents contain code and can exhibit intelligence; both the infrastructure and the agents must be able to protect themselves and their interactions.

References

1. W. Rosenberry et al. Understanding DCE. O'Reilly & Associates, Inc., 1992
2. Object Management Group. Common Object Request Broker: Architecture and Specifications (Document no. 91.12.1), 1991
3. Intelligent Agents, Communications of the ACM, 37(7), Special Issue, July 1994
4. C.G. Harrison, D. Chess, A. Kershenbaum. Mobile agents: are they a good idea ?, IBM technical report, IBM T.J. Watson Research Center, Yorktown Heights, NY, March 1995
5. D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, G. Tsudik. Itinerant agents for mobile computing, IEEE Personal Communications Magazine, October 1995
6. W. Joosen, F. Matthijs, J. Van Oeyen, B. Robben, S. Bijnens, P. Verbaeten. CORRELATE: High-level support for traveling agents, Technical report 236, Department of Computer Science, K.U.Leuven, 1996
7. J. Ousterhout. Tcl and the Tk toolkit, Addison-Wesley, 1994
8. L. Wall, R. Schwartz. Programming Perl. O'Reilly & Associates, 1990
9. The Java Language: A white paper, Sunsoft Inc., 1995
10. E. Jul, H. Levy, N. Hutchinson, A. Black. Fine-grained mobility in the Emerald system, Proceedings of the 11th ACM SIGOPS Conference, November 1987
11. W. Joosen, Y. Berbers, M. Snyers, P. Verbaeten. Transparent object migration in adaptive parallel applications, Proceedings of the European Workshop on Parallel Computing, March 1992
12. S. Bijnens, W. Joosen, P. Verbaeten. A reflective invocation scheme to realize advanced object management. In *Object Based Distributed Programming*, R. Guerraoui, O. Nierstrasz, M. Riveill (eds.), LNCS 791, Springer Verlag, 1994
13. F. Matthijs, W. Joosen, J. Van Oeyen, B. Robben, P. Verbaeten. Mobile active objects in CORRELATE, Proceedings of ECOOP'95 Workshop on Mobility and Replication, Eds. B. Anderson, C. Baquero, R. Oliveira, pp. 142-154, 1995
14. D. Johansen, R. van Renesse, F. Schneider. Operating system support for mobile agents. Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS), pp. 42-45, May 1995
15. J. White, Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, 1994
16. R. Gray. Agent Tcl: A transportable agent system. Proceedings of the CIKM Workshop on Intelligent Information Agents, Baltimore, Maryland, December 1995
17. J. Levy, J. Ousterhout. A Safe Tcl toolkit for electronic meeting places. Proceedings of the First USENIX Workshop on Electronic Commerce, pp. 133-135, July 1995