Motivation and Requirements for the AgentSpace: A Framework for Developing Agent Programming Systems

Alberto Silva
Alberto.Silva@inesc.pt
IST/INESC

M. Mira da Silva
mms@dmat.uevora.pt
University of Evora

José Delgado
Jose.Delgado@inesc.pt
IST/INESC

Abstract

A framework for supporting agent programming systems, called *AgentSpace*, is proposed in this paper. Our goal is to define an abstract and generic world of agents that is a super-set of research and industrial proposals currently being developed. There are mainly two novelties in this paper. First, we argue that real-world complex agent-based applications span through three levels: framework, programming system and application level itself. The second novelty is that, unlike many other proposals, our programming model focus on heterogeneous multi-agent applications.

Keywords: AgentSpace, Agent Systems, Agent-based Applications, Internet

1. Introduction

This paper concentrates on the development of distributed applications based on agents and their support programming systems. A *software agent* [WJ95,Nwa96] is a program that acts on behalf of a user to perform a very specific—usually repetitive and laborious—task. The user is usually a person, such as a manager in a company, that needs to accomplish the task delegated to the agent. Central to the paper is the idea that one agent is already useful in a computer; however, it can only take advantage of all its potential when there are multiple agents co-existing in what we will call *an agent community*.

This community of agents, unimaginable even a few years ago, can now be supported by the Internet. Thus it comes as no surprise the proliferation of proposals for *agent systems* that help application programmers write agent-based applications. Possible targets for these applications can be found in the following areas: personal assistants, workflow and collaborative applications, electronic commerce, and network management systems [MRK96, HCK95]. Many other areas will surely appear if these initial areas see a number of successful agent-based applications being used to solve real-world problems.

However, despite the increasing popularity of agents and their support programming systems, there is a consistent shortage of useful agent applications. For example, in the ECOOP'96 Workshop on Mobile Object Systems [BTV96] there was not a single paper describing such an application. Except in the conclusion, there were also very few discussions, if any, of possible applications based on agents.

This paper starts with a concrete multi-agent application as a motivation exercise and then goes on to propose a complete *agent framework* as a basis to support independent *agent programming systems* that can support this kind of application, as well as to design other agent-based applications (see figure 1).

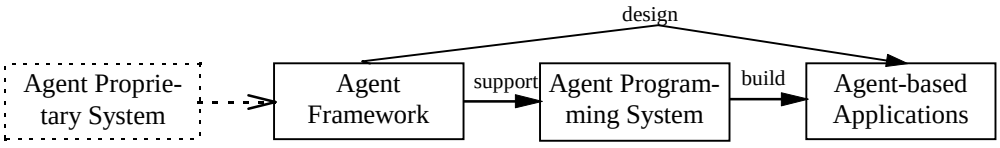

Figure 1. From Framework to Agent-based Applications

The motivation is followed by an overview of *AgentSpace*, our proposed framework for agent programming systems. The goal is to define an abstract and generic world of agents with the following characteristics.

- To provide an environment to discuss agents, agent systems, and applications.
- To propose an analysis and evaluation reference model as a first step towards a future integration and inter-operation between heterogeneous agent systems.
- To support the design of agent-based applications independently of their own proprietary agent systems.

There are mainly two novelties in the paper. First, the proposal goes through all three conceptual levels—framework, programming system and application—contributing towards a new global understanding of the agent research area. The second novelty is that we focus on *heterogeneous multi-agent applications.* Many agents that need to interact bring new issues such as agent communication, external services and resource access that in our opinion have not been sufficiently explored in other research work. Additionally, heterogeneity involves interoperability and integration issues between agents and between their corresponding (proprietary) agent systems.

2. Example Application

Although the current Web technology is already adequate to support the development of many distributed applications, it will not be adequate to support the kind of world-wide application that will be necessary in the future. This problem is illustrated below with a real large-scale distributed application that we are currently developing.

2.1 The Virtual Enterprise Network

The *Virtual Enterprise Network* (VEN) application is an Internet-based distributed information system that will support the cooperation between SME (small- and medium-size enterprises) in Portugal. Namely, VEN will maintain information about each company participating in the project, help their managers collaborate more and better, and in the future support remote training.

One of the services that VEN will provide is to support research projects being pursued by a number of these companies in consortium. They have to find the strengths and weaknesses of other companies, form alliances, write joint research proposals and then develop the project together.

A project is divided in a number of specific tasks. One of the companies is elected as the *coordinator* and becomes responsible for the whole project. The coordinator proposes who does what, negotiates with all *partners* the tasks to be accomplished, makes sure progress is being made, prepares the final demonstration and edits the technical reports. This implies a non-trivial set of tasks and coordination amongst disparate entities during the whole life cycle of the research project.

2.2 VEN Characteristics

The VEN application has a number of characteristics that by themselves have been dealt with independently in the past. It is their combination that poses problems.

- *Autonomous*—Each company creates and maintains their own applications using their own resources. (The last thing managers want is another manager from another company telling them what to do.)
- *Heterogeneous*—Each company has bought, got used to and uses different interfaces, machine architectures, programming languages, database systems, communication packages, operating systems and so on. They also have different programmers with different backgrounds and levels of expertise.
- *Open*—Some services may depend on other applications and even external organizations, thus the VEN has to inter-operate with other (legacy) information systems: applications, databases and so on.
- *Dynamic*—Applications will be added, updated and removed at any time without previous notice. The VEN applications will have to cope with unavailability, new interfaces, oscillating bandwidths, etc.
- *Robust*—The VEN will have to tolerate different kinds of failures on machines, networks or at any level of software. For example, the application cannot stop executing just because a company is rebooting their gateway to the outside world.
- *Secure*—The system should provide different levels of security depending on each particular part of the whole application. There will be public, VEN-specific and administrative applications and data.

VEN is only an example of a large class of distributed applications that will be developed on the Internet. Cardelli even gave a name to them: *global applications* [Car96]. However, it should be noted that not all of these global applications will be global in the world-wide sense; an application geographically distributed over Portugal may have the same basic characteristics as truly global applications.

3. The Web Approach

The current state-of-the-art of Web information systems—based on server-centric combined with client-centric technologies—may already support the development of some global applications such as VEN. Examples of technologies that could be used for VEN include: HTTP, HTML, CGI, Java, relational databases, ODBC and CORBA—all well-known and tested in the-real world. However, they present some limitations and deficiencies that we have described elsewhere [SBD95,SMdSD97]. In the following section we show briefly why these technologies cannot be used to support VEN.

3.1 VEN as a Web Application

Figure 2 presents a (simplistic version of a) possible *decentralized solution* based on current Web technology. Each organization provides its own Web-based applications, eventually with connections to their own legacy applications.

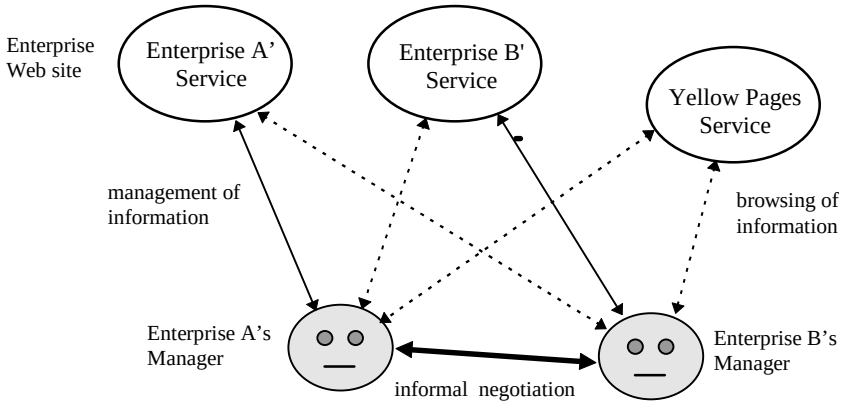

Figure 2: The *Virtual Enterprise Network* based on Web technology

Using the Web, managers are responsible for creating and maintaining their own Web sites with information about their companies' skills, products and services. In addition, if the manager is the coordinator of some project, then he or she has the responsibility of finding out the best set of available partners. In this case, the manager retrieves from the Web information on the best partners available (e.g., using a yellow pages service maintained by somebody else). When a good set of candidates is found, the manager then uses another technology (such as phone, fax, or e-mail) to negotiate directly with the other manager.

Another possible solution requires the existence of one special organization (or a restricted set of organizations) responsible by the development and maintenance of some *centralized service* in a single computer accessible to all managers. In addition to the yellow pages service—that tracks all members belonging to the VEN—this centralized service could provide all the desired activities (such as collaborative project establishment and management, search and retrieval of appropriate partners, and so on). However, a centralized solution restricts the autonomy and flexibility of each enterprise and cannot even use their applications if these already exist.

3.2 Current Web Technology Is Not Enough

Between the two solutions to implement the VEN described above, we still opt for a Web approach, although it presents some drawbacks. The manager must learn to use several applications, with different user-interfaces and eventually different interaction metaphors. Each application has to provide its own security and authentication mechanisms. The manager has to learn how to interact with all of them and keep a private list of authentication codes.

The manager in enterprise A, for example, has to maintain its own Web site, deal with B's Web site, search the yellow pages and talk to the manager of enterprise B. All these interactions have a different communication protocol. The problem grows quadratically to the number of companies in the application because all managers have to learn how to deal with all other companies! This is why the managers in figure 2 are not very happy.

Another issue concerns the synchronous mode of modern end-user interaction. Ideally, some tasks could be made asynchronously because of their isolated nature, low bandwidth, high-latency networks or just because they are too complex or too long. An example of such a task would be to match a set of requirements from a manager against a set of skills from all potential partners. A human interaction may take a long time (especially over lunch) and is prone to misunderstanding (especially after lunch).

On the other hand, if an automatic procedure was possible, then the answer could be found in the background by the VEN application itself. The current Web technology does not support this behavior very well because it is based on human-computer interaction, not computer-computer interaction that is the realm of agents.

4. The Agent Approach

In order to support the development of Internet-based distributed applications like VEN, several proposals in the area of *agents programming systems* have been made. Examples of these systems include: Telescript [Whi94], PageSpace [CTV96], Cardelli's Obliq [Car95a], Agent TCL [Gra95], Sun's Java applets [AG96], Mole [SBH96], IBM's Aglets [IBM96] and Persistent Java [AJDS96,ADJ+96].

4.1 VEN as an Agent Application

Figure 3 depicts (again, a simplistic view of) the VEN application now based on agents. The managers (end-users) are now happy because they only need to understand and interact with a single application with a single interface: the agent. This is even more remarkable because the agent with which they have to interact is their own agent, meaning that it can be configured to a particular user or set of users. For example, we can envisage very patient agents with colorful interfaces for computer illiterate managers, agents with effective (shell-based) interfaces for knowledgeable managers, or even agents that adapt to their managers dynamically (it all depends on the effort put on developing these agents).

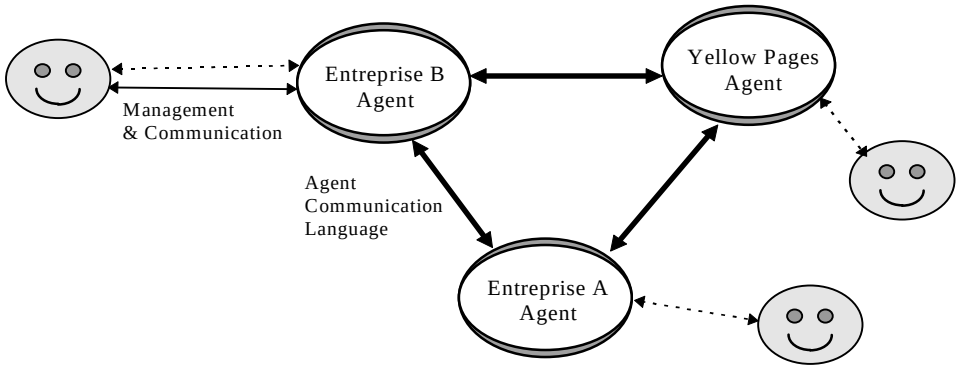

Figure 3: The *Virtual Enterprise Network* based on agents

This novel agent-based solution presents, when compared to the approaches described in the previous section, the following advantages:

- It is *decentralized* because the only centralized point is the yellow pages agent and *scalable* since this congestion point can always be attenuated or even eliminated by the incremental introduction of more yellow pages.
- It is *dynamic* because all agents can, with more or less flexibility, enter or leave the system. Additionally, the functionalities and complexity of the various agents may evolve independently.

- It promotes the *autonomy* and *flexibility* of each enterprise because they now become responsible for the development and maintenance of their own applications and particularly of their own agents.
- It offers a *single interface* because it hides the disparate complexities and user-interfaces of each application. As a consequence, the end-user only needs to interact directly with its own customized agent and eventually a few specific (management) agents.

4.2 Preliminary Issues

Although apparently an agent-based approach seems to offer an ideal development environment for VEN, there are some important drawbacks that will be described below.

In dynamic and open environments such as those we propose, *it is difficult to define and promote common agent-based application protocols and APIs*. All or the main involved organizations need to agree on a *common agent protocol* to allow agents to communicate. However, this protocol is specific to this application and so does not need to comply with any existing standard—usually, a restricted number of pioneer organizations design and agree a common protocol, then all other organizations just accept and adopt it.

Also, due to the non-existence of experience, models, techniques, tools and environments there is nowadays a *great difficulty for designing and building* really agent-based applications. The opposite is the database world: their RAD systems make it extremely easy to develop client-server SQL-based applications from scratch. In the agent world we are still many years away from having such tools.

Finally, several research and commercial agent systems are emerging, each with its own *proprietary agent model*. In fact, the majority of proposed agent programming systems are still lacking some technological aspects. For instance, Telescript provides a complex, persistent and secure system. However, its object-oriented (proprietary) programming language is very difficult to learn and use. On the other hand, Tacoma and PageSpace provide high level scripting languages (Tcl and Java respectively) that are more "natural" to learn and to use. However, these two systems do not fully address security and robustness issues when compared with Telescript.

There are several other issues whose merits are not well understood yet. For instance, what really is an agent? How is it created? When and by whom? What behavior can it present in generic applications? After being created and launched, agents then interact directly with each other on behalf of their managers. However, should the agent that interacts with its end-user be the same agent that also interacts with other agents? When the agent moves to another machine, can its end-user still interact with it?

All these difficulties and requirements would be even worse if the involved agents were based on distinct systems and technologies. Or, in other words, how should we address the heterogeneity issue? One elementary question should be, for instance: can a Telescript agent interact with an Aglet agent? If yes, how and to what extent?

5. Proposed Framework

This section, the core of the paper, gives an overview of *AgentSpace*, our proposed framework for supporting agent programming systems.

5.1 Agents, Nodes and Clusters

The *agent* is the basic entity of the framework. It executes some specific tasks on behalf of someone (a person) or something (an organization, another agent, etc).

The *AgentSpace* is the set of all agents and all *run-time environments*, that is, their corresponding computational (software) infra-structures where they execute (see below).

A *node* is a machine (hardware) infra-structure on which a computational infra-structure can be installed. These days a node is likely to be a computer, but in the near future it will probably include PDAs (now called "hand-held PCs"), TV sets, and mobile phones.

Nodes are logically organized in *clusters*. For example in the Internet context, all computers with an Internet name belonging to the domain "inesc.pt" belong to the same cluster. Clusters form a hierarchy in order to give some organizational and management functionality to the entire agent application. Usually a cluster suggests a geographical proximity, but it does not need to be so. An intranet, for example, can be implemented as a cluster. Depending on the application requirements, there may exist specialized agents for managing nodes and clusters called *system agents* (see below).

5.2 Agent Execution System

The agent run-time environment, provided in every node, is called AES—for *Agent Execution System*[1]. The AES can be built from scratch or, more likely, as a combination of existing hardware infrastructures (e.g., ordinary PCs), operating systems (e.g., Windows or JavaOS), communications packages (e.g., TCP/IP, HTTP or CORBA), and some kind of virtual machine (e.g., Java VM).

The AES provides a full computational environment to execute the agent, as well as other support mechanisms such as agent persistence, security and mobility. In order to support distributed applications and in particular agent mobility, different AES should communicate amongst themselves using some kind of low-level (read simple) protocols and agree in some common agent representation formats (e.g., using well-known marshaling techniques found in RPC systems). The AES should also provide or at least integrate specific APIs to allow access to external services and resources, such as databases, the file system and physical devices.

Furthermore, an AES should provide, in each node, one or more agent *execution places.*[2] These execution places are locations where agents execute and meet other agents. Every place is identified univocally by some electronic address. The implementation details of places vary and depend on each AES[3].

We agree with D. Chess et al. [CGH+95] in that, due to their complexity and inherent distributed characteristics, an AES should be better designed and built in terms of object services and supported by an existing distributed computing infra-structure such as DCOM or CORBA. In this way, several task of the AES can be delegated to its subjacent distributed infrastructure instead of "re-inventing the wheel" yet again.

5.3 Agent Communities

The basis for any *agent-based application* being built using our proposed framework is the notion of community. A *community* is formed by a set of agents that share their knowledge and communicate between them using a common language.

Knowledge is a description of some fact, some relationship between facts and/or other relationships in some restricted contexts. The knowledge maintained by one agent can be used by any other agent in the same community. (It is likely that at the implementation level the knowledge of all agents belonging to the same community will be maintained by a database, potentially distributed by a number of nodes.)

Communities are dynamic; new agents can enter the community at any time—if they are allowed by the community—or leave it. There should be protection against infiltration by foreign agents, perhaps based on real-world mechanisms: an identity card will be issued by the community, there will be friend communities, and so on.

A community is a logical concept that can be spread over a number of nodes or clusters. An agent belongs to one or several communities if it is allowed into those communities and can speak their languages. There will be *guest agents* that are allowed only limited access to the community knowledge, probably introduced by another agent that is responsible for its behavior. Other agents could be specialized as *translators* for different languages, *mediators* to resolve conflicts or *police agents* to stop or kill agents with bad behavior.

---

1 The AES is usually called agent system (AS). We prefer AES because AS has a too general use and sometimes vague or prone to misunderstandings. Some authors also call the AES an "engine" , "agent server", or "agent meeting point".

2 Some authors just call them "places", "locations" or "meeting points".

3 For instance, in Telescript a place is a process which can contain an arbitrary number of other places. This kind of places corresponds to an execution place and a stationary agent (that may provide several services) in our framework. In Mole, the location of a place is based on information of the IP address, port number associated with a Mole's engine, and a serial number.

Communities may be open or restricted. In the open case, all their agents (except the system agents) have similar characteristics and functionalities, and may work anonymously or not. In the restricted case, there should be different types of agents, identified (i.e., with some special tickets, permits or credentials) with different access levels and respective capabilities and available resources.

5.4 AgentSpace

As described in the previous section, a community is a set of agent-based applications sharing the same context and supported by a common AES. The inverse is not true. This means that the set of all agents supported by the same AES does not define, just by its own, a community.

The AgentSpace notion is an evolution relatively to the community concept in terms of desired capabilities and complexity. It is a dynamic set of agent-based applications related in same restricted context, but supported by different AES. This means that an AgentSpace needs interaction and communication between heterogeneous agents.

Figure 4 shows the relationships between agent, community and AgentSpace. The agent presents basic capabilities, such as autonomy, persistence, mobility, and communication with its user. A group of homogeneous agents related in a common context (i.e., sharing the same AES) defines a community, which is the second level of the hierarchy.

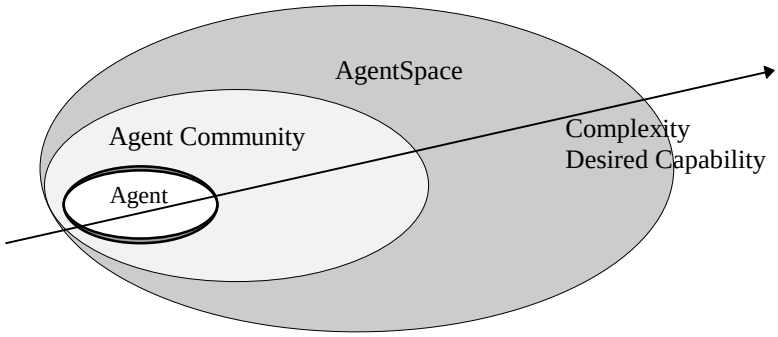

Figure 4: Hierarchical relations between agent, community and AgentSpace.

The community raises two new aspects. The first involves the need for a communication language between homogeneous agents. Basically there are currently two approaches: declarative (e.g., KQML) vs. procedural and/or object-oriented (e.g., Tcl, Telescript, and Java). The second aspect involves how to represent the specific knowledge of the community. There are also two basic approaches: knowledge representation languages (e.g., KIF and EDI) and specific APIs and protocols agreed amongst the principal entities responsible by the development and management of the involved community.

In the third level of the hierarchy, AgentSpace extends the community concept by allowing the communication and interaction between heterogeneous agents. Obviously, agent communication should be independent of any language or AES. In contrast to KQML (the declarative approach) we propose a *common interface language* for interacting between heterogeneous agents. The characteristics found on CORBA IDL make this protocol a good starting point. A current effort on that direction can be found in [TF96].

Other issue that requires in depth investigation concerns the notion of the AgentSpace delimitation. This issue raises several other questions, such as: does it make sense to define an AgentSpace of AgentSpaces? This means, does it make sense to define complex operations, such as aggregation and composition, around the AgentSpace concept? So, in the positive case, we may talk about an *open and universal AgentSpace*, where every agent may interact, subject to necessaries restrictions, with any other.

5.5 More on Agents

The agent can exist outside a community but will only progress towards achieving its task after it is accepted in one of the existing communities (or start its own!). Once accepted by a community, an agent can then interacts with other agents; for example, to ask a question, negotiate a deal, provide a service, advertise a service, sell a product, buy raw materials, or simply help other agents achieve their tasks.

Although we don't intend, at this specification level, detail agent internal implementation aspects, it is important to refer that, they should present a minimum set of well-defined characteristics: *identification* (i.e., name, electronic address and passport information relatively to their associated AgentSpace); *internal state representation* (i.e., code, specific data; common attributes, and execution image); and *external interfaces* (i.e., end-user interfaces, and agent knowledge- and services-based interfaces).

There are two basic kinds of agents: mobile and stationary (or static) agents. In general, *stationary agents* are created in the context of a very specific application at the user's initiative and become attached to that user during a long time period. Since these agents do not move, they do not present security problems to the system. However they should prevent other third agent attacks.

On the other hand, *mobile agents* are usually created by stationary agents and by other mobile agents. They are typically used to solve small and specific tasks and consequently have a short life. For example, the stationary agent of enterprise A, responsible for starting the project, may create a mobile agent to look for three potential partners for a task. This agent is launched by its parent to the yellow pages agent. There, it asks for electronic addresses of possible candidates, then it jumps to the execution places of the different stationary agents, meets and talks to them, and retrieves the information it needs. Finally, it comes back home and gives the information found to its owner (the stationary agent).

Finally, in applications like VEN mobile agents will make intensive use of sophisticated databases and carry data with them [MdSA96,MdS96]. Others have agreed that this integration between code and data will be crucial to the development of agent-based applications [Whi94,BC95,ADJ+96]. The reader is referred to the main issues involved with integrating persistence and mobility in a companion paper [MdSS97].

5.6 End-Users

All *users* of the agent application have at least one agent that executes some tasks on their behalf. Users give their agents specific tasks or brokerage and mediation capabilities. *User agents* are agents owned by end users that behave as "consumers" of services. The information consumed by user agents is "produced" by *application agents*, for example, those that manage databases of products on sale. Both of them are usually stationary agents but may occasionally create, at run-time, mobile agents to execute very specific tasks.

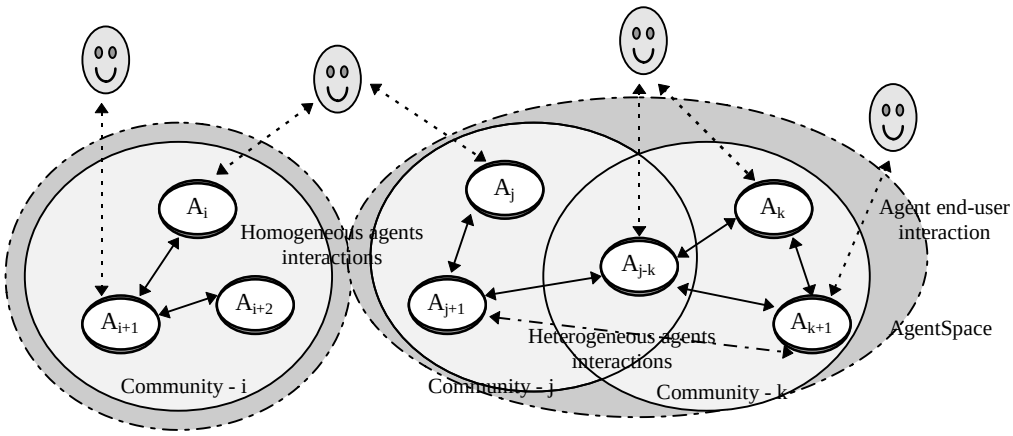

Figure 5: AgentSpaces and agent interaction patterns.

Users may interact with their agents in several ways depending on their interface characteristics. For instance, they may interact through e-mail messages, HTML forms, AWT-base Java applets, Active-X components, etc. (Ideally, this interaction should be voice-based in natural language!). They have all rights over their agents, namely to suspend, change their knowledge and goals, or even eliminate them. Nevertheless, these rights are restricted by the supported AES as well as by the political rules of the involved AgentSpace.

6. Conclusions and Future Work

The paper presented VEN, an example agent-based application, and how it can be implemented both by using the current Web technology and an agent programming system. We concluded that only agents can effectively support VEN, although current agent programming systems lack the functionalities needed to implement VEN.

We then introduced the main concepts and requirements of our framework for developing agent programming systems that will support better communication amongst agents and integration between code and data. These can then be used to implement the kind of application exemplified by VEN.

As part of our future research work we will develop a prototype VEN application using an existing agent programming system (probably the Aglet workbench from IBM or Persistent Java from the University of Glasgow). We will propose an independent agent development model, with which we will discuss two new emerging themes: how to design agent-based applications? and what tools and components should agent programming systems provide in order to simplify their development?

References

[ADJ+96] M.P. Atkinson, L. Daynès, M.J. Jordan, T. Printezis and S. Spence. An orthogonally persistent Java. *SIGMOD Record*. 1996.

[AG96] K. Arnold, J. Gosling. *The Java Series - The Java Programming Language*. Addison-Wesley Publishing, 1996.

[AJDS96] M.P. Atkinson, M. Jordan, L. Daynès and S. Spence. Design issues for Persistent Java: A type-safe, object-oriented, orthogonally persistent system. In *Proceedings of The Seventh International Workshop on Persistent Object Systems* (Cape May, New Jersey, USA, May 29-31, 1996). Morgan Kaufmann Publishers, 1996.

[BC95] K. Bharat and L. Cardelli. Migratory applications. *Proceedings of the ACM Symposium on User Interface Software and Technology* 1995 (Pittsburgh, PA, Nov 1995). 1995.

[BTV96] J. Baumann, C. Tschudin and J. Vitek. *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems* (Linz, Austria, July 8-9, 1996). Dpunkt. 1996.

[Car95] Luca Cardelli. A language with distributed scope. *Computing Systems*. 8(1):27—59. Jan 1995. (A preliminary version appeared in Proceedings of the 22nd ACM Symposium on Principles of Programming Languages.)

[Car96] L. Cardelli. Global Computation. *Position Paper*. 1996.

[CGH+95] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris. Itenerant Agents for Mobile Computing. *IEEE Personal Communications*. 2(5):34-49, Oct. 1995.

[CTV96] Paolo Ciancarini, Robert Tolksdorf, Fabio Vitali. PageSpace: An architecture to coordinate distribute applications on the web. *Computer Networks and ISDN Systems*. 28, 941-952, 1996.

[Gra95] R. Gray. *Agent Tcl: a transportable agent system*. Proceedings of the CIKM Workshop on Intelligent Information Agents, (CIKM'95), 1995.

[HCK95] C. Harrison, D. Chess, A. Kershenbaum. *Mobile Agents: Are they a good idea?*. IBM, 1995.

[IBM96] IBM Tokyo Research Laboratory. *Aglets workbench: Programming mobile agents in Java*. http://www.trl.ibm.co.jp/aglets. 1996.

[MdS96] M. Mira da Silva. *Models of higher-order, type-safe, distributed computation over autonomous persistent object stores*. PhD Thesis, University of Glasgow. (Submitted in December 1996.)

[MdSA96] M. Mira da Silva and M. Atkinson. Combining mobile agents with persistent systems: opportunities and challenges. In [*BTV96*].

[MdSS97] M. Mira da Silva and A. Silva. Insisting on Persistent Mobile Agent Systems with an Example Application Area. Accepted to the *First International Workshop on Mobile Agents*, 1997.

[MRK96] T. Magedanz, K. Rothermel and S. Krause. Intelligent agents: an emerging technology for next generation telecommunications. *Proceedings of INFOCOM'96*, San Francisco, USA, 1996.

[Nwa96] H. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*. 11(3), 1-40. Cambridge University Press, 1996.

[SBD95] A. Silva, J. Borbinha and J. Delgado. Organizational management system in a heterogeneous environment: a WWW case study. *Proceedings of the IFIP working conference on information systems development for decentralized organizations* (Trondheim, Norway, August 1995). Pages 84—99. 1995.

[SBH96] M. Strasser, J. Baumann and F. Hohl. Mole: A Java-based mobile object system. In [*BTV96*].

[SMdSD97] A. Silva, M. Mira da Silva, J. Delgado. A Survey of Web Information Systems. Submitted to the *WWW'97 Conference*. 1997.

[TF96] C. Tham, B. Friedman. Common Agent Platform Architecture. *General Magic*, Oct. 1996.

[Whi94] James White. Telescript technology: The foundation for the electronic marketplace. *General Magic White Paper*, General Magic, 1994.

[WJ95] M. Wooldridge, N. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*. 10(2), 115-152. Cambridge University Press, 1995.