Building a Collaborative Information Agent in a HomeBot Application

Lito Perez Cruz, Arkady Zaslavsky, and David Squire

Monash University, Faculty of Information Technology, School of Computer Science and Software Engieering, PO BOX 197, Caulfield East, Victoria 3145, Australia
lito,Arkady.Zaslavsky, David.Squire@csse.monash.edu.au,
WWW home page: http://nemesis.csse.monash.edu.au/~lcruz

Abstract Information agents are cooperative agents whose main goal is to collaborate and assist task agents like mobile agents in obtaining information from heterogeneous data sources. Considering the current tools and techniques available in the community, this study sets out to find the practical problems that a developer might encounter in the process of engineering an information agent in the context of a real estate agent HomeBot application. In this technology assessment report, we provide here the approach for circumventing them and document the lessons we have learned. Our assessment is that the construction of an information agent based on current agent building technologies is feasible but not seamless. More importantly, we contend that what is needed is a wholistic tool that integrates smoothly various sub-technologies together. This is not yet provided for in the community and something agent tool builders should address immediately.

1 Introduction - The Problem of Data Access

With the wider acceptance of the Internet, intelligent agents and in particular mobile agents, reach their maximum potential when they are allowed the ability to interrogate various data sources present in the network. In general, there are two general data sources, namely: web pages or databases. This is not a trivial problem to solve, as some would assume.

Putting aside for the moment, the problem of security and the issue of reasoning capabilities, there are two strategies that the mobile agent may employ. Firstly, it may access the HTML pages that pertain to its domain. The other is to access the database where the actual data is found. Both of these present complicated problems for the mobile agent

For a web-based approach, the mobile agent must be capable of collecting the web based information in a manner that is systematic and orderly since web pages are in raw text format and in general un-structured. This is the reason that researchers looking at this problem apply artificial intelligence techniques such as natural language processing to transform the text and dump it into a database for efficient processing [8]. Some like [9] use information agents to do this, but in our view this technique is time consuming as it has to dump the site onto a local computer first and only then is it analyzed. This

means also that in practice, the dynamic nature of the searching process is hampered. Changes in the web page content and format are not easily accounted for.

Assuming a consortium of cooperating web sites exists for a particular domain of application, the more efficient technique is to make the databases they hold the target of data sourcing. Besides, web page information come from databases anyway. Complication is diminished if an intelligent agent, like a mobile agent, is allowed direct access to these databases in the first instance. Simply letting the mobile agents read the real estate database is the path of least technical resistance. For this reason, we pursued the database access approach due to its more practical advantages. However, there is a problem, how can the mobile agent access the database if it does not know the names of the tables and fields it contains? How can it issue query statements that are sensible to the database being interrogated? In view of this, an information agent that accesses these varied databases on behalf of a mobile agent becomes the ideal bridge for assisting task (mobile) agents achieve their goals. A sample problem that the mobile agent is trying to solve is found in Figure 1

In this study, using real estate (RE) property buying/selling as domain, we uncover the practical issues of developing an information agent (IA) using current available tools and techniques. Our study is, in effect, a technology assessment report on the maturity of current agent construction tools as applied in an RE environment. The context is a scenario wherein a mobile agent is commissioned to search for a ”best property fit” on behalf of a prospective buyer (in this case a human agent). It is characterized by a mobile agent visiting one real estate agent site after another with the hope of obtaining the property that best fulfill the requirements of a user. When a mobile agent reaches its candidate site, it negotiates with the resident IA of that site since it is the IA that knows how to access that site’s real estate property database. The IA assists the mobile agent in accessing data down to the database level through some mapping support. The mobile agent then proceeds to visit the next site in the list once the ”best fit” is obtained from the current site. The operation terminates when all candidates sites have been visited and then the best of the “best fit” is evaluated. This is depicted in Figure 2.

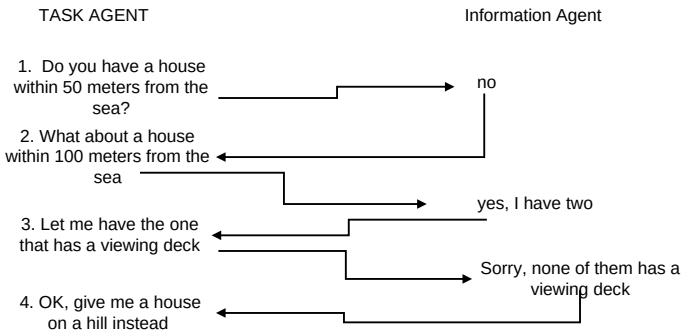

Figure 1. Sample Task and Information Agent Interaction

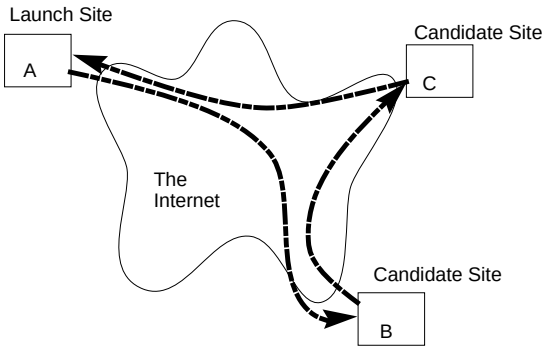

Figure 2. Mobile agent starts from A and visits B and C

We approached the issue from the following angles:

1. *Break down the IA into its functional components or parts*
2. *Discover the technologies that address those functional components*

In the next sections we discuss the experience of actually implementing an information agent based on agent tools available in the community.

2 Finding an Architectural Model

An information agent (IA)is a special case of an agent. Its task is to support other agents in retrieving data sources. The main question is how to define in detail the role the IA is to play. This will dictate aspects of what architectural model is suitable to use. Because of this, whatever architectural issues are present in other agents are applicable to information agents, except for mobility issues since they are designed to be static and situated.

A few points are worth noting. In [2] one finds a paradigm for architecting in a high level way an agent. For example, it needs perceptors and acceptors and so forth. At a minimum the IA must *perceive -reason-act* [1]. This model is of course applicable to the information agent. In [2] there is a notion of programming an agent in a ”dynamic” way, in that the information agent is designed to be aware of some state information external to the information agent. That is, the information agent should be made aware of its ”environment” by way of keeping track of certain useful states of processing. This state information is lodged in a knowledge base (KB). We believe this is a useful concept that should be included in the programming of an information agent as this enhances its resiliency.

In addition to the high level model of [2], the example of [4] is highly applicable. In the work of McKay, Pastor, McEntire and Finin [4], one will see that they developed a KB with its representation using the LOOM Knowledge Representation Language which is a derivative of KL-One. The application program talks to the KB to get common

information and then talks to an Information Agent called LIM (or Loom Interface Module). LIM on the one hand takes messages coming from the application program which are in the form of a representation language embedded inside KQML statements. We use this idea to enhance what we already have with some slight variation.
In our case we replace the KB with an ontology (discussed further below). The information agent is given the task of obtaining the request for information coming from the mobile agent. The mobile agent speaks to the IA singularly without necessarily accessing the KB. This is slightly different from [4]where in the client agent has access to the KB. In our case we are using the IA as a wrapper for all support services that the mobile agent requires in relation to raw data access and retrieval. This is found in Figure 3. It is presumed that prior to launching the mobile agent is already in possession of a search goal and that the IA will service the request. Determining this scope boundary is an issue and it is best to simplify the role of IA in that we limit its function to that of satisfying the search request. To summarize we have broken down the functional components of our IA and depicted them in Figure 4. The middle parts such as the reasoning part and the security part are at the moment left out of discussion for the following reasons; a.) there are various reasoning support and paradigm that can be chosen from and b.) security issues in this context are a major topic that is best left as separate discussion on its own.

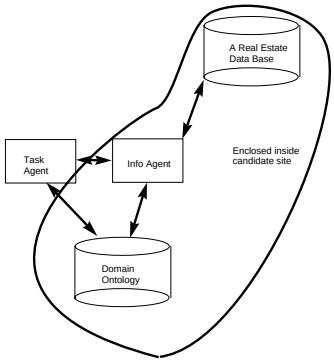

Figure 3. KB replaced by an Ontology

3 Suitable Agent Communication Language

We mentioned that once the mobile agent reaches the site, it them talks to the information agent that is resident in that site. There should be a way for the mobile agent to communicate with its information agent so that the request can be passed along, thus a communication mechanism is a recommended need. Firstly the language should be humanly readable and secondly, it should be high level enough to be intuitive. To develop a message protocol for agent communication will divert the developer's attention

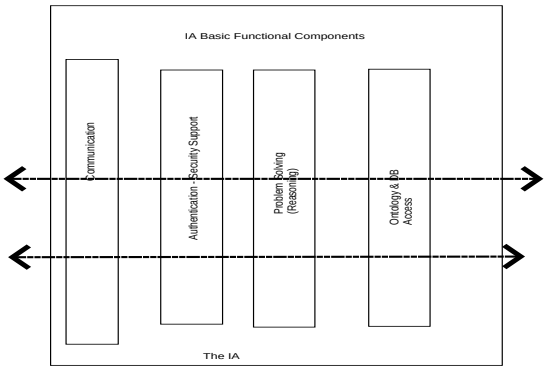

Figure 4. Our IA's Functional Components

to the real task so, an adoption of existing agent communication language is more interesting and is more prudent. There are two agent communication languages that are available for consideration, namely KQML or FIPA ACL. A comparative analysis of this is found in [3]. Which of these are chosen will dictate the direction as to what tools are readily supportive of the choice. At the time of the project it was KQML that was chosen, but because the FIPA ACL has lots of commonality with KQML, we believe in the future that FIPA ACL should be considered strongly due to other supporting infrastructure that is gathering around FIPA compliant agents. The point is that the choice of communication language will dictate the agent tool to use.

4 Use of Ontologies

How can we pick the right real estate property from databases whose schema is different, whose platform is different (for example Oracle DB vs mySQL vs Sybase ASE vs postgeSQL etc) and whose table names for example are different? As an example, sales-price in one DB may be represented as list-price in another DB. To resolve this would not be possible with out a concept map that translates a real estate concept to actual terminologies used by the databases. A structure called an ontology fulfills that gap [10]. The ontology acts as a concept map so that the task agent may be able to deal in some universal way with real state cites whose schema is assumed to be non-uniform in every cite. Is there already an existing real state ontology that we can use? The answer is 'yes'. The Real Estate Transaction Standards Working Group (RETS)(http://www.rets-wg.org), has such an ontology. Their standard is embodied in a document called Document Type Definition (RETS DTD). RETS DTD is a workable starting point for creating an ontology. The document is not an ontology as such as in that it is not in a format acceptable to a KR language. It is in XML format. It is at least a start. The task then is to translate the presentation into some KR language format like DAML (or DAML+OIL)[5]. A commitment to an ontology language of course needs to be predetermined in the process of constructing an information agent. This determi-

nation also dictates the provisions of a tool. Is the chosen ontology language supported by the tools?

5 Construction Tools

Agent technology research is at least a decade old and many organizations are producing tools that will make agent programming as easy and as productive as possible. Briefly, tools may be classified in terms of license, they may be commercial or academic. In terms of philosophy, they may be classified either as a framework or as an agent development environment(ADE) similar to that of Rapid Application Development (RAD)tools. Frameworks are building blocks and offer a lot of flexibility but the disadvantage is that more work has to be done by the programmer. ADEs provide good gains in productivity but are not as flexible as the frameworks. They tend to lock the user to a particular component. For example, a tool may support KQML and not FIPA-ACL. An example of commercial tool and also an ADE might be AgentBuilder [11] and JACK [12]. An example of an academic product and an ADE is ZEUS [13]. An example of an academic product and a framework is JADE[14]. In our work we experimented with these tools and at the same time tested with Voyager[15] and Grasshopper[16] as platforms for agent mobility. We were interested in finding out based on our architectural considerations, which tool was able to give us maximum leverage and productivity. We experimented with a few of the mentioned tools putting emphasis on ease of implementation or construction. In the next section we discuss our findings.

6 Lessons Learned

6.1 Architecture - This goes first

The engineering of an IA is no different from the development process undertaken in developing a generic agent. The normal process of starting at the requirements level then proceeding to design, development and so forth is still applicable. The first step was to identify architecturally one's functional components. This dictates which tool provides such functionality support. The functional model strongly dictates the type of technology to use. The task was to identify which tool best delivers all or most of our architectural components. We assert that another developer whose view differed from our functional decomposition will arrive at a different construction approach. Our view however is that, an IA should have the following supported technologies.

1. Problem solving or reasoning capabilities
2. Agent communication languages
3. Ontology and DB access support
4. Authentication facility

These architectural components should be prioritized as to importance because it is not prudent to assume that there is a tool that addresses all architectural requirements. The prioritization will help alleviate any conflicts between tools that supply a requirement

but also misses on another. In our experience one may choose a tool with good reasoning support but may not support the communication language one requires. Such cases were in abundance. We believe that the above functional components may be thought of as services. The idea is to weave together these 'services' into an orchestrated whole. However, at the present writing they have not been turned to services yet, but with the advent of web services, it may not be an absurd idea to expect these components one day to be agent services that one may avail of.

6.2 Communication Language- Who supports it?

At the moment there are a few tools that support directly agent communication languages like KQML but those that support FIPA-ACL are quite scarce as well. We find that the decision to choose one over the other will require the programmer to lose some advantages given by tools that provide other ideal functionalities. This is where the prioritization of components come into play. Even if at present the programmer wishes to use a particular agent communication language but there is no ready support for it in the tools, then that would be fruitless. As an experiment we used a tool that supported KQML. How did we use it then? The task agent and the IA passed KQML messages through a supported API call. Because most tools are Java based, it was a matter of creating a type of *KqmlMessage* object and listening on it through a type of *percept* object. We made the communication happen through a 'port' in the host site.

6.3 Ontology - What type?

Present construction tools view ontologies as plug-ins that must be included or injected into the scenario. Should one commit to an ontology language and its corresponding engine? In one of the tools we experimented, it was possible to implement an ontology by use of a class object that is constructed at run-time from the RETS DTD.
It came as a surprise that there are building tools that made no use of logic based ontologies (also known as general purpose ontologies) at all. Indeed, AgentBuilder and Zeus do not advocate one. These do mention an ontology, but they implement it simply as a Java class object(also known as domain-specific ontologies). The argument for the latter is its simplicity . This raises an important question: how important is it for a HomeBot application to rely on a general-purpose ontology? Is it helpful at all for our case? The answer depends on how much reasoning sophistication is required. Because our aim was immediate deployment we opted for a domain-specific ontology. We justified this because the HomeBot and the IA would usually belong to the same consortium of RE companies. The benefit was that the scenario became simpler and the need to map description logic statements to query language such as SQL was avoided. The style is to wrap DB and ontology objects inside a Java class. To query the ontology, one simply goes through that Java class' methods. We note that KQML or FIPA-ACL allow you to insert a transport command like SQL inside your agent communication statements. With this method we were able to insert our desired SQL that is passed from the task agent onto the IA. The principle is still the same even if a general purpose ontology had been employed. It just takes more time programming. The reason is that, in both cases,

ontology access would have been performed through some ontology access class/object mechanism. The plus side is in its power.

6.4 The technology gap

Our functional decomposition of what parts an IA should have dictated the required tools we needed to avail of and combine together. Our first comment then is that there is no one single agent building tool that combines all into one package and we wish there was one. This means that there is a void here that can be filled by an insightful agent-building tool company. What we mean here is that each has facilities that are lacking that others may have. For example, Voyager allows for mobile agent migration, but it does not have an IDE like AgentBuilder or Zeus for creating visually an agent. On the other hand AgentBuilder has no mobile agent facility. JAFMAS [6] and JACKAL [7] has KQML support but no mobility like Grasshopper. JAFMAS and JACKAL are frameworks i.e. they give you the skeleton for creating agents with no supplied default behaviors so you can be as sophisticated as you want your agent to be (but more work). Zeus on the other hand has almost all features except a mobile agent facility although this is planned. In the end, we found the combining of the facilities of AgentBuilder[11] with Grasshopper [16] in building a HomeBot and IA application a smoother experience as a whole. Note that the trick is to combine the said facilities into a holistic system, but this presents a problem in programming. The developer has to be creative in programming to work around the idiosyncratic tastes of these tools. As an example, these various components make use of API calls, we discovered that in attempting to weave these third party tools together as a whole, we ran into sociability problems. For example, the APIs names clashed. Other times a fix to one idiosyncrasy became a fault on the other and caused our IA to be unstable. Moreover, because the developer becomes like an integrator, obtaining support for programming questions becomes a no-man's land. Some of these tools are GPL and the suppliers are likely to ignore pleas for help if a mixture of products is involved in the agent program. Supplier finger pointing is a possible scenario. This can be avoided if a comprehensive tool or environment is present. An opportunity exists therefore in this area. To summarize then, the ideal agent building tool should combine all features into one package having namely, the following features:

1. Mobile agent facility and tracking
2. IDE for visual agent creation and debugging
3. KQML or FIPA-ACL support
4. Ontology support
5. Reasoning engine
6. Communications framework
7. Object Brokerage like CORBA

7 Conclusion

Our aim was to discover at once the practical issues that a developer might face and deal with them immediately. Thus, broad issues like security and choice of reasoning style

are reserved for another report. We highlight that the way we decompose the functions of an IA plays a vital role in determining the components needed to build an IA. Our decomposition was intuitively based on agreed properties that IAs should have. Our resulting list of functions determined the facilities the tools should provide for the construction. The properties must be prioritized as to importance. This helps choose one tool over the other when inadequacies in the technology emerge. At the present, there is no single tool that is integrated such that a programmer can do a one-stop shop on a tool that delivers all typical properties. The tools are flexible enough to allow for the creativity but they requie the hard work of plugging in the functionality, this will affect the implementation schedule. However, more effort implies more project risks. They therefore, make the implementation exercise longer to finish. There is an opportunity for a tools vendor to deliver such a tool, one that provides a mobility framework, an ACL support, a pluggable ontology as well as reasoning service and so forth. There are products that are close to becoming like that (for example Zeus which only lacks mobility infrastructure). Overall, the agent developer stands to benefit if there is an available tool that integrates all the component properties together in a seamless whole. We hope that this wish becomes a reality soon.

References

1. Lind, Jurgen: Issues in Agent-Oriented Software Engineering. AOSE (2000) 45–58
2. Russell, S and Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall Series in Artificial Intelligence (1995).
3. Labrouu, Y., Finin, T., and Peng, Y: Agent Communication Languages- The Current Landscape. IEEE Intelligent Systems, V.14,2. (March/April 1999).
4. McKay, D., Pastor, J., McEntire, R., and Finin, T.: An architecture for Information Agents. Advanced Planning Technology (ed) A. Tate, AAAI Press, Menlo Park, CA., USA,(1996).
5. Gomez-Perez, A., Corcho, O.: Ontology Languages for the Semantic Web. IEEE Intelligent Systems (January/February 2002).
6. Graham, J. and Decker, K.: Towards a Distributed, EnvironmentCentered Agent Framework. In: N. Jennings and Y. LeSperance (eds.): Intelligent Agents VI, Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99). Lecture Notes in Articial Intelligence. Springer-Verlag, Berlin.”,(2000).
7. Cost, R. S., Finin, T., Labrou, Y., Luan, X., Peng, Y., Soboroff, I., Mayfield, J., and Boughanam, A.: Jackal: A Java-based Tool for Agent Development. AAAI-98, Workshop on Tools for Agent Development,Madison, WI, (1998).
8. Soderland, S.: Learning to Extract Text-Based Information from the World Wide Web, In Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97), 1997.
9. Ga, X.: A methodology for building information agents. Intelligent Agent Laboratory, Department of Computer Science, The University of Melbourne, Australia
10. Hendler, J: Agents and the Semantic Web IEEE Intelligent Systems, vol.16, no. 2, Mar./Apr. (2001) pp. 30–37
11. Reticular Systems Inc. http://www.agentbuilder.com
12. Agent Oriented Software Group. http://www.agent-software.com
13. BT Exact Technologies http://193.113.209.147/projects/agents.htm
14. CSELT S.p.A., University of Parma. http://sharon.cselt.it/projects/jade
15. Recursion Software Inc. http://www.recursionsw.com/
16. IKV++ http://www.grasshopper.de/index.html