Autonomous Knowledge Agents
How Agents use the Tool Command Language

Raymond W. Johnson

Artificial Intelligence Center
Lockheed Missiles and Space Corporation
3251 Hanover Street
Palo Alto, CA 94304-1191

Abstract

The Autonomous Knowledge Agent project is an internal research and development project being conducted at Lockheed's Artificial Intelligence Center. The goal of the project is to develop an architecture for creating and running personalized software agents. This paper will discuss what traits we require of a software agent. We will then discuss how one requirement, that an agent must be a first class citizen, led us to use Tcl as a command language for agents and the services and tools they use.

Introduction

Computers are so prevalent today and information so abundant that computer users are being barraged with too much information. More and more time is being spent by people searching for and retrieving information. A plethora of simple tasks such as updating a common directory of files or sending out frequent mailings also add to computer overload. The net result is that more and more people are becoming slaves to their computers. Agents hold the potential promise of relief from the overload of information.

The agent metaphor is derived from the idea of a personal assistant (in this case a Personal Digital Assistant or PDA). Much like a secretary, the agent (acting as a PDA) is delegated simple and/or periodic tasks. Ideally an agent should be able to do any computer related task you may delegate to a secretary. However, given the current state of artificial intelligence technology, it's only practical to build agents that can handle tasks in restricted domains.

The architecture we have designed allows for the management of most any type of agent. An agent has the potential to run any type of code and use any type of service. All that is specified is a high level control protocol that allows agents to be queried for their status, to be activated or deactivated, or to be given other messages defined for that a specific agent. Other services such as scheduling and communication links are also available to the agents but they are not required to use them.

However, in this broad architecture agents are as difficult to design and create as writing traditional programming code. To aid in the creation of agents an engineer may design a "domain template" that will allow agents in a given domain to be created by users with no programming experience. The domain template consists of knowledge of the domain, the code required to carry out potential tasks, and a graphical interface through which the user specifies the desired agent behavior. The use of domain templates make the creation of agents simple without restricting the power of agents in other domains.

What makes an Agent?

We mention above that our architecture can support an agent as defined by any piece of code. However, an agent must posses certain behaviors and abilities to be considered a PDA and not just a another programmable tool. We believe agents should have the following characteristics:

• Agents should be Autonomous. They must be able to run without user intervention. Other than being told what to do, they should appear to have a life of their own.

• Agents should be Intelligent. They should be able to figure out *how* to carry out *what* the user has requested and should be able to accommodate a myriad of potential events and situations. They can not expect to ask for help from the user when problems arise.

• Agents must be first class citizens. By this we mean that agents must be able to potentially do any computer related task a human user may do. An agent should be able to control or drive any other application or service on the machine.

Each of these characteristics present interesting and challenging problems. In this paper we will limit the discussion to the problem and possible solution of making agents first class citizens.

The struggle for equality

The current state of most computer environments is strongly biased against programmatic control as required by software agents. The graphical user interface (GUI) revolution created many applications that only accept control via mouse clicks or movement and dialog box style keyboard input. Control of such applications by other programs is virtually impossible. Even "command line" environments such as UNIX provide inconsistent access to the control of applications.

To alleviate this inequality for agents all applications and services must be controllable in a consistent programmatic way. One solution is to define a scripting language that every application must support. This has the side effect of making services and tools independent of their user interfaces. Agents may then use services and tools in a more "natural" way (to software agents, anyway).

The difficulty is, of course, that even if a dictatorial law demanded that every application be scriptable it would still take years for the thousands of existing applications to support scripting. Until such time, however, agents will remain second class citizens.

The Role of Tcl

Fortunately, several operating system environments have taken the step to evangelize the benefits of scriptability and have provided tools for supporting it. Most notably among these HP's NewWave environment (HP89) and Apple's forth coming Apple Script (MW93). However, neither of these scripting systems met all of the requirements we had for our agent project.

It was decided that the Tool Command Language (TCL) or "tickle" should be used for our project. It had the advantage of being non-proprietary and not very system dependent (although it does have a strong UNIX bias). Tcl is also extremely extensible and very powerful. Finally, the issue of portability played heavily into our choice of using Tcl.

The main use of Tcl was to provide a consistent interface to the various services and tools our agents would use. For example, Tcl interfaces were defined for WAIS[1] database services, SMTP[2] mail services, and the CLIPS[3] knowledge base system. An abstract view of the use of Tcl in our architecture is given below.

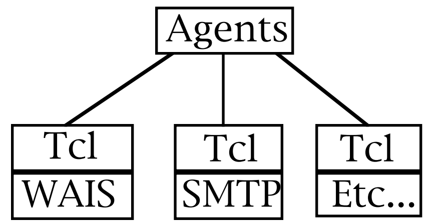

Fig 1. The use of Tcl

This has several advantages that reduce the complexity of a software agent. Instead of writing Tcp/Ip sockets for WAIS accesses, creating apple events to mail messages, and calling C functions to access the knowledge base an agent only needs to use Tcl procedures to access all services and tools. Having a consistent language for all services makes the creation of agents a much simpler task. In addition, because the implementation of services are independent from the Tcl interface, the interface may stay the same over various implementations. This has made it possible for agents designed on a Macintosh to run with little or no changes on a UNIX workstation.

---

[1]WAIS - Wide Area Information Services
[2]SMTP - Simple Mail Transfer Protocol. It is used to send internet mail.
[3]CLIPS - C Language Integrated Production System. CLIPS is a product of NASA's Software Technology Branch.

Tcl is also used to control the agents themselves, in fact at the highest level agents are simply Tcl objects.[4] However, the object itself may do nothing more than pass messages on to the actual agent. An agent of a given domain is represented as a class that must be inherited from a basic agent super class. This basic class defines the core messages any agent must be able to handle, for example, the message asking for an agent's status. The use of Tcl objects give us a way to hide and encapsulate the definition of an agent and provide a consistent way to communicate with them.

One item Tcl lacks is a consistent protocol for sending Tcl scripts to other applications. Most UNIX X11 applications use the Tk "send" command, while many Macintosh applications (including ours) use the Do Script apple event. We are currently using different communication protocols for sending to different applications. While we are planning to develop a more consistent Tcp/Ip based communication scheme, we hope some cross platform standard for sending scripts will emerge. Fortunately, we have been able to abstract the communications protocol to a level where a change in the protocol will not require a change to the agents.

[4]OBJECT Tcl is a Tcl extension written by Lockheed's Artificial Intelligence Center. It is not an official (or supported) component of the Tcl language.

Example Agent Domain for Managers

Every department within Lockheed writes weekly activity reports that describe work in progress and new undertakings. It is important for managers to keep abreast of these reports to avoid wasting resources by "reinventing the wheel." The reports are also important for managers who need to find other groups within Lockheed to team with or share technology. Assuming the contents of these reports were compiled and made available in an electronic format, we could define an agent domain we'll call the "weekly activity report" domain. The use of agents in this domain has the potential to save Lockheed from wasting valuable resources.

Agents in the "weekly activity report" domain have the role of helping managers keep abreast of the deluge of activity reports. All the manager needs to do is enter keywords about the domain she is interested in (Boolean logic may also be used). The manager may also specify how the information should be reported back to her. For example, the agent may send interesting activity reports to the manager via a graphical interface, electronic mail, or even a Fax. The agent, being savvy about the domain, may perform formatting, collation, or other processing without the user having to specify such actions

When creating an agent, the most common options for this domain are easily defined via a simple graphical user interface. Advanced behavior can be defined in a rule based manner or by supplying Tcl scripts. However, because of the limited domain and the ease of the GUI, writing scripts will rarely be necessary.

Future Work

Once a domain template like the "weekly activity report" domain is created, developing and managing agents is fairly simple. Currently, the creation of these domains is a difficult and time consuming task. The long term goal of our project is to make the creation of these domains much easier.

The creation of a domain involves creation of the high and low level actions an agent my perform, developing the knowledge reasoning ability of the agent in that domain, and finally a user interface that allows the agent to be easily assembled by a non-technical user. Of these, the most time consuming is the creation of the user interface.

The windowing tool kit Tk, which is built on top of Tcl, is ideal for creating simple interfaces and requires much less coding than traditional windowing tool kits. We plan to incorporate Tk when it is ported to the Macintosh. We believe Tk would offer us two major advantages. First, we would have cross platform compatibility of the user interface. In addition to agents, we desire the ability to be able to transfer domain templates across platforms without extensive modification. Second, we feel Tk is an ideal medium for automatically creating user interfaces from domain knowledge. Several systems have demonstrated the ability to automatically create a user interface for an application in a small limited domain (Foley91)(Wiecha90). We believe that the limited domains of agents and the relatively low complexity of Tk scripts, make it feasible to automatically create a Tk interface for an agent domain based on the knowledge and actions defined for that domain.

Conclusions

All in all we have found Tcl to be an indispensable part of various facets of the AKA project. Its flexibility and extendibility has allowed it to serve several important roles. As a tool command language, Tcl provides us a consistent interface to the tools and services an agent may use. As a scripting language, Tcl provides a simple way to define high level tasks and agent actions built upon the basic tools and services. With the help of the object oriented extensions, Tcl was also used as an agent control language and created a simple way to communicate with agents.

Tcl is still young and we hope for its continued success. We also hope that Tcl will become widely used on many operating systems (including a port of Tk to the Macintosh and Windows) and resist becoming too tied to the UNIX operating system.

Acknowledgments

The research behind this paper is funded from an internal research and development project sponsored by Lockheed's Artificial Intelligence Center in Palo Alto. The author would like to thank Chris Toomey for his guidance and help with editing this paper.

References

HP89 Stearns, Glenn R. Agents and the HP NewWave application programmer interface. Hewlett-Packard Journal v40, n4 (August, 1989) p. 32.

MW93 Cohen, Raines. AppleScript Toolkit, runtime version will ship this month. MacWEEK vol. 7 num. 14 (April 5 1993) p. 1

Foley91 Foley, J., Kim, W., Kovacevic, S., & Murry, K., Defining Interfaces at a High Level of Abstraction, IEEE Software, Jan. '89, pp. 25-32

Wiecha90 Wiecha, C. & Boies, S. Generating User Interfaces: Principles and use of ITS Style Rules, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology: UIST '90. Snowbird, UT. Oct. 3-5, 1990. pp. 21-30.