Comparison of Expert System Shells via Testbeds

Umur Ozkul
Levent Akin
Ethem Alpaydin
Ufuk Caglayan
Selahattin Kuru

Department of Computer Engineering
Bogazici University

e-mail: ozkul@boun.edu.tr

June 8, 1994

Abstract

The potential mismatch between benchmark scores and performance on real tasks leads us to use testbeds to compare and evaluate expert system shells. In this paper we describe our experience in testbed agent design, and our experience in integrating a testbed, Truckworld and expert system shells.

Introduction

Expert system shells[[3]] are software packages containing a generic inference engine, a user interface, and a collection of other tools that enable users to develop and use expert systems. Using the shell a knowledge engineer can create and modify knowledge bases. The inference engine contains inference and control paradigms. The interface allows users to interact with the shell.
The process of evaluating expert system shells is difficult, and is not standardized. Benchmarks are a simple and useful method of evaluation. Nevertheless, the potential mismatch between benchmark scores and performance on real tasks leads us to use testbeds to evaluate expert system shells.
In this paper, we describe our experience of testbed agent design using expert system shells. We extended Truckworld[[1]] so that it is easily integrated with expert system shells.

Truckworld

The Truckworld[[1]] is a multi-agent testbed designed to test theories of reactive execution. The main commitment is to provide a realistic world for its agents. An agent is a truck consisting of two arms, two cargo bays, several sensors, and various other components like a fuel tank, a set of tires, and direction and speed controllers. It operates in a world of roads and locations. Objects populate locations connected by roads. Exogenous events like rainstorms occur periodically in the world. The Truckworld also contains facilities for communication and other interaction among agents.

Integration with the Truckworld

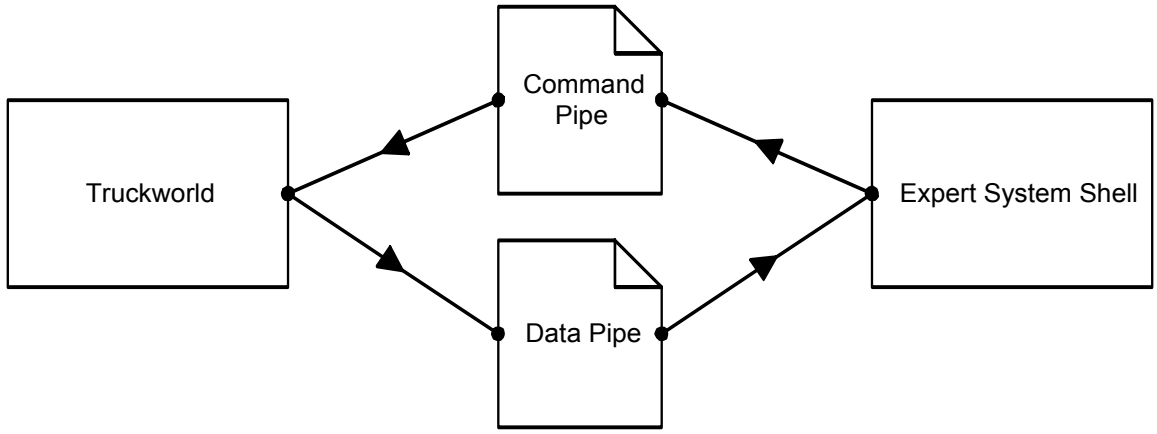

Figure 1 Integration of Truckworld with an expert system shell

We have implemented an additional module in Truckworld. This module allows the Truckworld to use UNIX pipes as illustrated in Figure 1. The “Command Pipe” carries truck commands from the shell to Truckworld. The “Data Pipe” carries sensor and simulation data from the Truckworld to the expert system shell.
You make things happen in the Truckworld by issuing commands to the truck. Commands to the truck generate events. If the command issued causes reading from a sensor, the data returned contains the objects seen through the sensor. Otherwise it is only the current simulation time.
A program accessing pipes cannot be distinguish them from ordinary files. So we just use file write operations to issue commands to the truck, and file read operations to fetch sensor data. In this manner, neither a special implementation nor an extension is required on behalf of the shell.

Agent Design

It is possible to design very complicated scenarios to go on in the Truckworld. For example, the trucks may be in a battle. Some of the trucks can cooperating to fulfill their job. Another example might be such that the trucks may be in a transportation and delivery job. On the other hand, we can design very simple scenarios. We can design a truck that simply travels the world and learns about the locations. We can create a truck that is curious about boxes. It temps to look into boxes whenever it finds one.
Whichever scenario we design, reactive planning comes into play in the Truckworld. The truck cannot hope to execute a series of movements without being interrupted or without changing its mind.
Let’s think of a simple case. The truck wants to grab an object it has seen. Just to plan and execute, imperatively, a series of arm commands is not enough in this case.
Suppose the truck, succeeded its movements and finally issued a command to grab the object. It might fail for the object is too heavy. This is the least of thing that can happen.

Just another truck can grab and carry away the object in the mean time. Then our truck will be stupidly moving its arm to grab nothing.
Finally our truck can have the virtue of changing its mind. In the mean time, the truck decides to do a more important thing. It might either totally forget grabbing or continue its movement later.
This is why we mention reactive planning to create a sensible truck. You cannot guarantee any plan to execute successfully. If you think this is not the case when you are the only truck in the Truckworld, exogenous events will take care of you. Of course, as we are writing truck agents simply to compare different expert shells, we might choose not to bother but applying reactive planning is a good way to force the features of a shell into realistic use.

Equivalence of Programs

As we are planning truck agents to compare and evaluate the performance of different expert system shells that we use to implement the truck agents, another problem emerges. The problem of portability of the

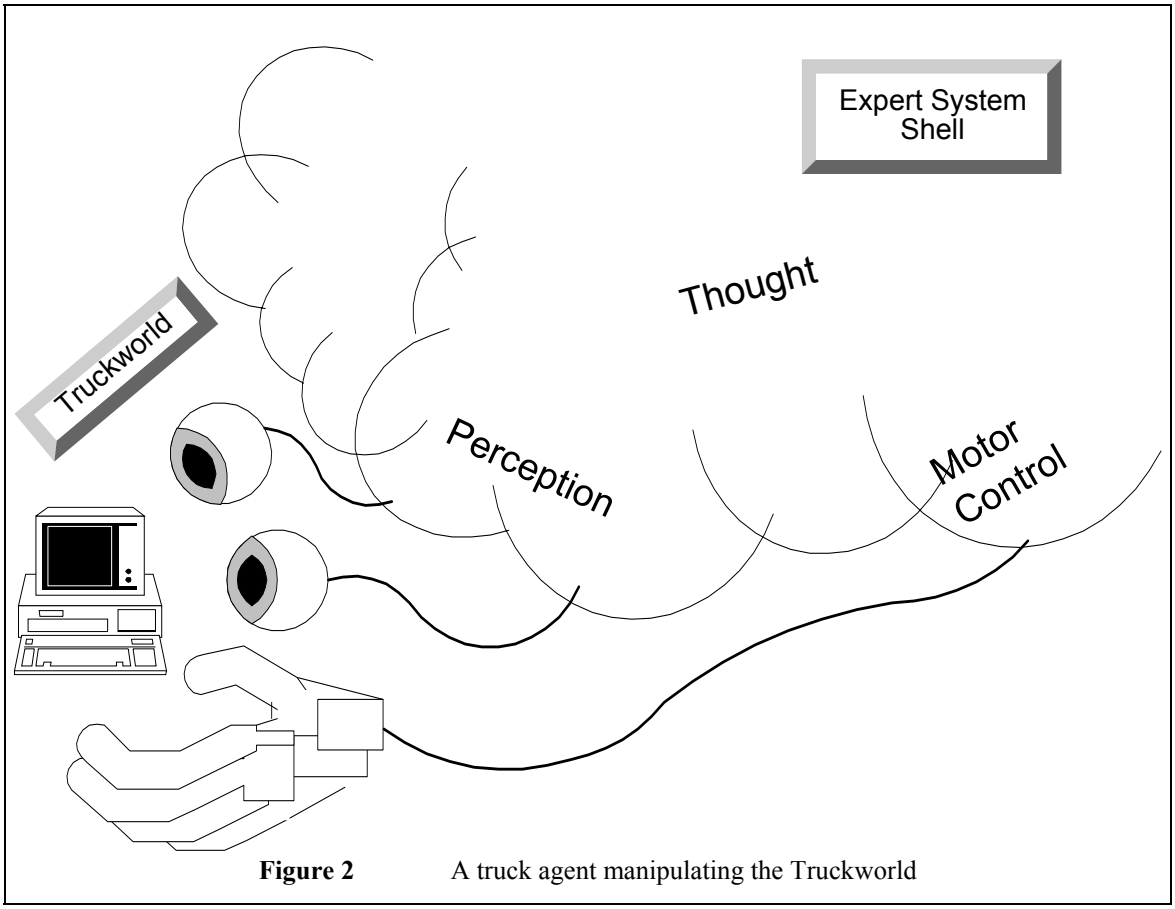

Figure 2 A truck agent manipulating the Truckworld

implementation of the agent.
To speak of a comparison, each implementation of a truck agent on each shell must be the best possible implementation. This is not possible. If we knew the answer we would know how to write a compiler or a translator that generates most optimal code.
Sometimes the problem of comparison is as severe as comparing a Prolog interpreter and a C compiler. C always beats Prolog in performance. So, can you discard Prolog? There are many problems that we will never attempt to code in C, and there are many Prolog codes such that the cost of converting then to C beats the lower performance cost.
Although the problem of comparing expert system shells does not seem to be that severe, expert system shell development is a fresh area, and they are coming into use with different new features.
Problems above said, it is worse in practice. Suppose you have two groups of freshmen students. The first group learns Prolog before learning Pascal. The second group learns Pascal before learning Prolog. It has been shown that the first group is more comfortable in Prolog programming. There is a negative learning transfer for the second group. Thus we cannot be sure if there is not a negative learning transfer between any pair of programming paradigms, facilities, etc. Consequently, if a set of programs is running a bit slower or harder to implement on a specific expert system shell, it might be simply due to our educational background.
Hence, we have to keep the programs simple and comparable while forcing a realistic use of the shells (So that we can see what we cannot see in benchmarks).

Agent Architecture

We assume reactive planning is a good place to start for something simple and comparable. Think of the arm movement example above.
Furthermore it seems that animals use similar brain architectures for arm movement. The same idea is applicable to truck agents although they have got different intelligence and manners.
From "bug-like" to "human-like" trucks we might have a somewhat reusable module to coordinate the basic movements of a truck. We suppose coding less also makes it more enjoyable to design a series of truck agents.
We perceive a truck agent as in Figure 2. In fact, there are three layers of code:

1. Interface Layer
2. Perception and Motor Control
3. Thought

Interface layer is where you make OS calls to interface with the outer world. Perception organizes data you collect. Motor Control coordinates basic primitive movements where a simple reactive planning proceeds. Thought layer is where you taste other AI hypothesis to create truck agents fulfilling a scenario.
Finally, we might judge and compare written using different shells more easily by modularizing the software.
Figure 3 illustrates the idea roughly in more detail. There a composite movement is a interruptable series of basic movements to satisfy a goal. In our case, basic movements are only commands given to the Truckworld. An example of composite movement is "grabbing an object". It is formed of basic arm movements of which the final one is grabbing. Self sense is a truck's information about itself. In fact this data is also obtained by the sensors of a truck. Of course not to waste time, a truck can guess about itself. For example, the position of its arms. If you want to grab an object, isn't it true that your next arm movement depends on the current position of your arm?

Conclusion

We have implemented an integration of a testbed, the Truckworld and expert system shells. We seek for a sensible comparison of shells by observing agents respective agents implemented on each of them.
One naturally tempts to use reactive planning in the Truckworld environment. Applying reactive planning is a good way to force the features of a shell into realistic use.
Also, if a set of programs is running a bit slower or harder to implement on a specific expert system shell, it might be simply due to our educational background as the implementors. Therefore we have to make it easy to judge about the code as far as possible. So it is wise to modularize the programs to obtain simple and comparable parts while designing realistic truck agents.
Furthermore it seems that animals use similar brain architectures for arm movement. The same idea is

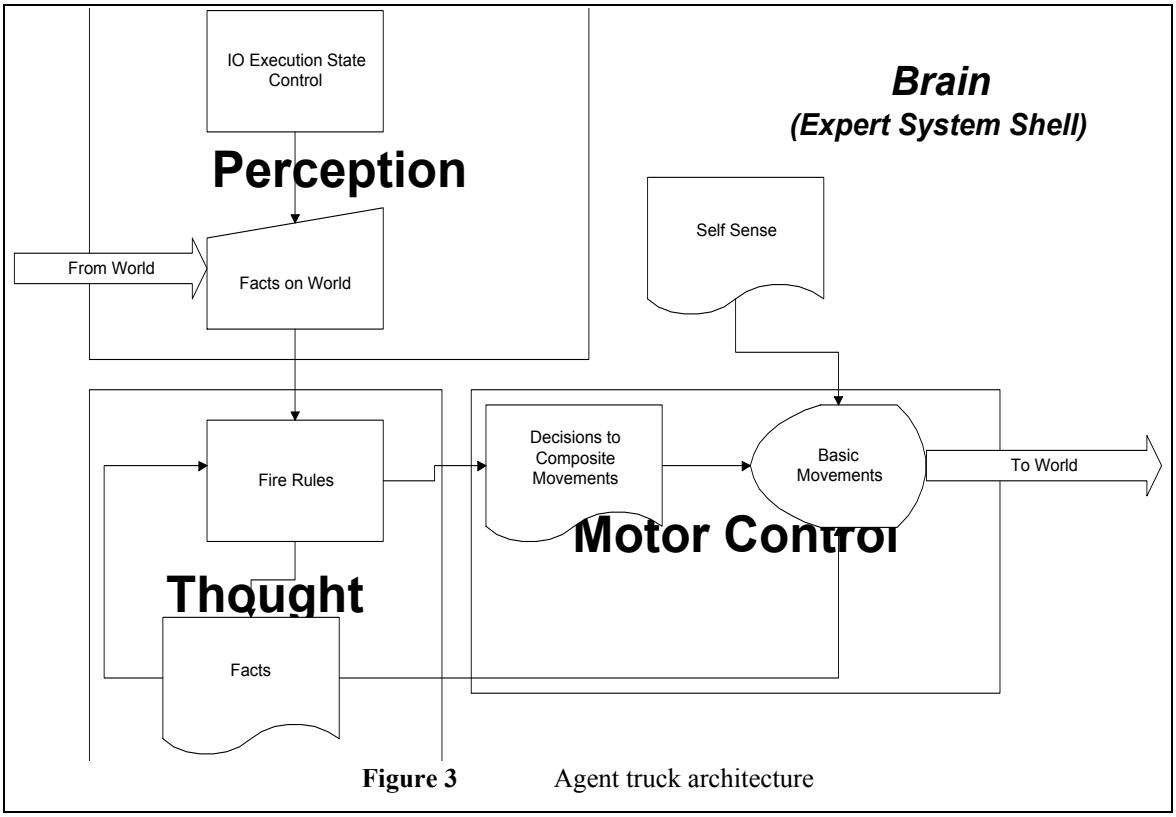

Figure 3 Agent truck architecture

applicable to truck agents although they have got different intelligence and manners.
The scientific value of well-crafted testbeds is there to highlight interesting aspects of system performance, but this value is realized only if the researcher can adequately explain why his or her system behaves the way it does.
Also it is not always obvious whether the lessons learned from the simplified systems are generally applicable, but neither is it obvious how to perform systematic experiments without the simplifications.

References

[1] Hanks, Steve, et al.; A Beginner's Guide to the Truckworld Simulator; Department of Computer Science and Engineering; University of Washington; Technical Report UW-CSE-TR 93-06-09 June 25, 1993
[2] Hanks, Steve, et al.; Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures; Department of Computer Science and Engineering, University of Washington; Technical Report UW-CSE-TR 93-06-05 June 17, 1993
[3] Stylianou, A. C., et al.; An Empirical Model for the Evaluation and Selection of Expert System Shells; The Belk College of Business Administration, University of North Carolina