Humphrys, Mark (2001), *The World-Wide-Mind: Draft Proposal*, Dublin City University, School of Computer Applications, Technical Report no. CA-0301.

The World-Wide-Mind:

Draft Proposal

Mark Humphrys
Dublin City University, School of Computer Applications,
Glasnevin, Dublin 9, Ireland.
tel (+353 1) 700-8059, fax (+353 1) 700-5442
humphrys@compapp.dcu.ie
http://www.compapp.dcu.ie/~humphrys/

This is a first draft of ideas for the WWM system, as at Feb 2001.
Further revisions will not appear here, but rather in a number of forthcoming papers.

Abstract

In the first part of this paper, a change in methodology for the future of AI and Adaptive Behavior research is proposed. It is proposed that researchers construct their agent minds and their agent worlds as *servers* on the Internet. 3rd parties will use these servers as components in larger systems. In this scheme, any user on the Internet will be able to (a) select multiple minds from different remote "mind servers", (b) select a remote "Action Selection server" to resolve the (inevitable) conflicts between these minds, and (c) run the resulting constructed "society of mind" in the *world* provided on another "world server". All this without *necessarily* having to consult with the server authors. This constructed society may now also be presented as just another primitive mind server, ready for reuse by others as a component in a larger system.

From the current situation of isolated experiments we will move to a situation where not only can researchers use each other's agent worlds, but they can also use each other's agent minds as components in larger systems. Servers may call other servers, and it is expected that 3rd parties will continuously write wrappers and filters for existing mind servers, overriding and modifying their default behaviour (to produce new, co-existing mind servers). None of this *necessarily* means that the mind being used ever leaves its server (or that its insides are even made public). Hence the term, the "World-Wide-Mind" (WWM), referring to the fact that the mind may be physically distributed across the world, with parts of the mind at different remote servers.

Part of the motivation for the WWM is that if the AI project is to be successful, it may be too big for any single laboratory to complete. So it will be necessary both to decentralise the work and to allow a massive and ongoing experiment with different combinations of components (so that we are not locked into any *particular* layout of decentralisation). Central to the WWM scheme is the expectation that researchers will *not* agree on how to divide up the AI work, and so components will overlap and be duplicated.

Previous work by this author [Humphrys, 1997] introduced models of mind where competition took place between extremely incompatible components, and where the mind could survive communications failure with or even complete loss of a number of such components. The WWM idea grew out of this work, and this paper shows how these previous models are the type of models we need in the WWM.

In the second part of this paper, we move towards an *implementation* of the WWM by trying to define the *set of queries and responses* that the servers should implement. Clients (including other servers) may then implement *any* general-purpose algorithm to drive the servers through repeated use of these queries. In our initial implementation, we consider schemes of very *low-bandwidth* communication. For instance, schemes where the competition among multiple minds is resolved across the network using numeric weights, rather than by explicit reasoning and negotiation. It is possible that this low-bandwidth protocol may be more suitable to *sub-symbolic* AI than to other branches of AI, and that other protocols may be needed for other branches of AI. It is suggested that it may be *premature* in some areas of AI to attempt to formulate a "mind network protocol", but that in the sub-symbolic domain it could at least be attempted now. Whether the protocol presented here is adopted or not, the first part of this paper (the *need* for a protocol) stands on its own.

Finally, we suggest a lowest-common-denominator approach to actually implementing these queries, so that current AI researchers have to learn almost nothing in order to put their servers online. As the lowest-common-denominator approach we suggest the transmission across ordinary CGI of queries and responses written in XML.

Keywords - World-Wide-Mind, WWM, Network Minds, Distributed Models of Mind, Society of Mind, Distributed AI, autonomous agent architectures, Action Selection, Internet, client-server, HTTP, CGI, XML, AIML.

Part 1 - Introduction

1 Introduction

For decades, AI researchers have constructed physical and simulated environments ("Worlds"), designed physical and simulated robots and agents ("Bodies") to interact with them, and designed, learnt or evolved behaviour-producing control systems ("Minds") to drive the Bodies. These Minds are typically used once, in a set of experiments, and then discarded. They can (in theory) be reconstructed from the details provided in the scientific papers, but in practice few ever see the light of day again. The papers, for good scientific reasons, concentrate on extracting the general *principles* behind the Minds, rather than on ensuring that the Minds' actual existence continues.

In nature, by contrast, minds are produced by the million, and get to live in millions of bodies throughout the world for millions of years. The contrast is dramatic with the testing and development of AI minds confined to a single laboratory, and often only a single "body", for only a year or two (during 90 percent of which the body is actually inactive). This paper suggests that the Internet provides a way for the field of AI to develop its own rich, world-wide, always-on, ecosystem to parallel the natural one.

First we ask what the problems are with the existing situation.

1.1 AI is too big a problem

The starting point for our motivation is the argument that the AI project is too big and complex for any single laboratory to do it all. Many authors have argued along these lines, and a number of different approaches to this issue have evolved:

- The traditional AI approach has been to work on *subsections* of the postulated mind, such as we find in computer vision, or language processing. In practice this has often meant working on specialist problems where it is assumed that *if* this system was built into a "whole mind", other modules would provide its input and also be able to process its output. And if the whole mind is never built, it does not matter because the system can stand on its own as a useful application, with *humans* providing the input and interpreting the output.

  The criticism of this approach is that the "whole mind" never actually gets built, and as a result many issues, such as the large-scale architecture of a complex multi-goal creature, and how it resolves its many internal conflicts, never get addressed. e.g. See the criticisms of [Brooks, 1986, Brooks, 1991]. For a symbolic AI call to build whole systems see [Nilsson, 1995].

- The *Animats* approach [Wilson, 1990] is to start with *simple whole creatures* and work up gradually to more complex whole creatures. This approach has generated much good work about overall architectures and conflict resolution. But as the complexity scales up, it cannot avoid the question of whether one lab can really do it all. Perhaps the Cog project [Brooks, 1997, Brooks et al., 1998] is now beginning to hit those limits.

- The *evolutionary* approach is to say that control systems are too hard to design and must be evolved [Harvey et al., 1992]. While it is true that this is how the most *advanced* control systems known arose, success so far in following this path has been limited. Extraordinary things have been evolved [Sims, 1994] but they are not, at least in the area of agent control systems, noticeably more advanced than the control systems that can be designed by hand or through learning.

  In any case, this paper shall not take a position on the merits of pure evolution v. design or learning. The methods proposed will be such that could be driven by evolution if so desired. For the moment we simply note that the evolutionary approach may also share with the animat approach an implicit assumption that one lab can do it all.

It seems to me that all these approaches still avoid the basic question, which is: **If the AI project is too big for any single laboratory to do it all, then as we scale up, how will we link the work of multiple laboratories?**

If the work in AI is to be distributed, with different laboratories engaging in specialisation, and then some scheme to combine their work, many problems immediately arise. Who decides who works on which piece? Will we spend all our time trying to define protocols to talk to other pieces? What if people can't agree on how to divide up the work, or indeed what the pieces are? [Brooks, 1991] Will this scheme force everyone to use a common programming language? Will it enforce a common AI methodology and exclude others?

Whatever scheme we invent cannot enforce any of these things, nor can it define a particular way of dividing up the work of AI. Rather it must be a *framework* for exploring different ways of dividing up the work, and different ways of combining heterogenous and overlapping modules from diverse sources. (By "overlapping", we mean that the modules have not *agreed* on who does what. There are some inputs for which more than one module will attempt to provide an answer.)

One thing we will need is a model of mind that can cope with multiple overlapping and conflicting minds in the same body, and will *assume* the existence of multiple unused minds at all times.

1.2 Duplication of Effort

Up until now, the above problems with schemes for sharing work have tended to scare researchers off, and instead everyone tends to invent their own system. In order to test a new learning algorithm, say, a researcher will first design his own simulated world, or his own robotic testworld, or even design his own robot. This takes a huge effort of time and work, and the problem world must be debugged and shown to be a hard problem, all before you even get to test your novel algorithm. How many animats researchers have spent (perhaps wasted?) their time designing their own gridworlds over the last number of years? Similarly, [Minsky, 1996] has criticised the recent "new AI" focus on using robots, saying researchers spend all their time learning robotics and get to do very little AI.

There are other problems, such as that designing your own world means your algorithm is not tested in the same world as previous algorithms. Machine Learning (ML) in particular has recognised this, and has attempted to define testbeds in which multiple algorithms can be tested and compared. See the datasets at the UCI ML Repository or MLnet [mlnet.org]. [Bryson et al., 2000] suggest the setting up of a website with similar standard tests for autonomous agents and adaptive behaviour.

There have been some attempts to re-use test agent minds [Sloman and Logan, 1999, Sutton and Santamaria, undated] and test worlds [Daniels, 1999], but they suffer from some of the same problems that have held back re-use [Humphrys, 1991] in the mainstream computer world - complex installation, and incompatibility problems with libraries, versions, languages and operating systems. Software is often easier to *re-write* than to re-use [Spolsky, 2000].

This paper suggests that for AI re-use we look at perhaps the most successful recent model of re-use (often not recognised as such) - the using of other people's documents and programs off remote Web servers. The Web has now made it commonplace for people to refer to data at a remote server rather than providing a copy of the data themselves. It has also made it commonplace for people to refer to the results of a program *run* on a remote server, rather than install the program themselves. We will suggest a similar model, where the agent test world essentially *stays* at the remote server and is used from there, instead of being copied.

This paper will not just address the problem of re-using other people's *worlds*, though. The fact that it is difficult to re-use other people's agent *minds* has also held back experimentation with complex, large-scale architectures.

1.3 Unused agents and worlds

Having invested the time and effort in inventing a robotic or agent testbed, few people then get to use it. Only MIT students and researchers get to run algorithms on Cog. This is at most a couple of dozen people. By nature's standards, Cog does not get to "live" much [Humphrys, 1997a]. There is only one instantiation of it,

and - like all similar AI projects - it will in practice be inactive 90 percent of the time. The world's AI labs are full of robots that have not been activated for years, awaiting funding (and *time*) for the next researcher to set up some brief experiment with them. Similarly, in the computers of the world's AI researchers is a whole history of unused virtual worlds and agent minds, stored in offline directories, awaiting a reactivation that may never come.

Putting these agents and their worlds on the network could maximise their utilisation and testing, in the same way that the original purpose of the Arpanet/Internet was to maximise the utilisation of rare and expensive remote federally-funded mainframes. While their creators have moved on and are busy with new projects, the agents and worlds will survive and will be linked to people who have the time and inclination to do further work with them. Experiments will be done with them that their creators would never have thought of.

Clearly, it will be easier to provide remote access to a *software-only* world (you can give every client their own copy) than to a *robotic* test world - the robot owners will want to control who (if anyone) uses their robots remotely. This will be addressed in detail below. For software-only worlds it should not be hard to set up a server to provide the existence of the world long after its creator has departed. For example, animat researchers would be very interested in the existence of a server for a complex animal behaviour world, such as Tyrrell's world [Tyrrell, 1993].

1.4 Minds will be too complex to be fully understood

To complete the criticism of the current situation, there is definitely something in the evolutionary criticism (referred to above) that advanced behavior control systems may be too complex to be *understood*.

As AI scales up, it is envisaged that minds will take on a level of complexity analogous to, say, a national *economy*, where each component may be understood by someone, but the dynamics of the entire system is too complex to be grasped by one individual. In the system we will propose, of a vast network of servers calling other servers, each individual *link* in the network will make sense to the person who set it up, but there is no need for the system as a whole to be grasped by any single individual.

Part 2 - The World-Wide-Mind

2 The World-Wide-Mind

The proposed scheme to address these issues is called the "World-Wide-Mind" (WWM). The name has a number of meanings, as will be discussed. In this scheme, it is proposed that researchers construct their agent minds and their agent worlds as *servers* on the Internet.

2.1 Types of servers

In the basic scheme, there are the following types of server:

1. A **World and Body server** together. This server can be queried for the current state of the world: `x` as detected by the body, and can be sent actions: `a` for the body to perform in the world.

2. A **Mind server**, which is a behavior-producing system, capable of suggesting an action: `a` given a particular input state: `x`. Note this does not mean it is stimulus-response. It may remember *all* previous states. It may take actions independent of the current state, according to its own independent plans. The Mind server may work by any AI methodology or algorithm and may contain within itself any degree of complexity. It may itself call other Mind servers. In the latter case we call it a **Mind$_M$ server** to reflect the

fact that it communicates with another server or servers (which may themselves communicate with many more servers) unknown to the client.

3. A special type of Mind server (in fact a special type of Mind_M server):

    1. An **Action Selection or AS server (or Mind_AS server)**, which resolves competition among multiple Mind servers. Each Mind server i suggests an action a_i to execute. The AS server produces a winning action a_k. This may be one of the suggested actions ("winner-take-all") or it may be a new, compromise action [PhD, §14.2].

        The client talks to the *AS server* to get an action, and the AS server talks to its list of Minds, using whatever queries it needs to resolve competition (perhaps repeated queries). Because it can produce an action a given a particular input state x, the AS server *is* in fact a type of Mind server itself. This is why we call it a "Mind_AS" server.

        But it may be a very different thing from the other Mind servers. The other servers actually try to solve some problem or pursue some goal. Whereas the Mind_AS server may be simply a generic *competition-resolution device*, that can be applied to *any* collection of Minds without needing any actual understanding of what the Minds are doing. We have 2 possibilities:
        1. The list of Minds is hard-coded into the Mind_AS server.
        2. The list of Minds is provided as an *argument* to the Mind_AS server at startup.

        In either case, typically all the actual *problem-solving* intelligence (suggestions of actions to execute) resides in the Minds.

It is imagined that each of these types of server may be customised with a number of parameters or arguments at startup, so that a client may ask for slightly different versions of what is essentially the same server. We have already seen possible arguments for a Mind_AS server (the list of Minds). Other possible server arguments will be discussed later.

2.2 Types of Societies

By allowing Mind servers call each other we can incrementally build up more and more complex hierarchies, networks or societies of mind. We will call any collection of more than one Mind server acting together a **Society**. A Society is built up in stages. At each stage, there is a single Mind server that serves as the interface to the whole Society:

1. A Mind_M server calls other Mind servers. To run this Society you talk to the Mind_M server.
2. A Mind_AS server adjudicates among multiple Mind servers. To run this Society you talk to the Mind_AS server.

and so on, recursively. At each stage, there is a single Mind server which serves as the interface to the Society, to which we send the state, and receive back an action. For example, in the simplest type of Society, the client talks to a single Mind and World server, who talk to no one else:

1. client talks to:
    1. Mind
    2. World

In a more complex society, the Mind server that the client talks to is itself talking to other Mind servers:

1. client talks to:
    1. Mind_M, which talks to:
        1. Mind
        2. Mind
    2. World

or:

1. client talks to:
    1. Mind_{AS}, which talks to:
        1. Mind
        2. Mind
        3. Mind
    2. World

and so on, with an endless number of combinations, e.g.:

1. client talks to:
    1. Mind_{M}, which talks to:
        1. Mind
        2. Mind_{M}, which talks to:
            1. Mind
        3. Mind_{AS}, which talks to:
            1. Mind
            2. Mind_{M}, which talks to:
                1. Mind
            3. Mind
    2. World_{W}, which talks to:
        1. World

(We will discuss "World_{W}" servers later.)

Each Society has precisely one Mind server at the top level. This makes sense because the competition must get resolved *somewhere*, so that an action is produced. And the *client* can't resolve it. So it must be resolved by the time it gets to the client.

2.3 Types of users

There are the following types of users of this system:

1. A non-technical **Client** user - essentially any user on the Internet. Basically, the client will run *other people's minds* in *other people's worlds*. Without needing any technical ability, the client will be able to do the following:

    1. **Pick one Mind server to run in one World**. Even this apparently simple choice may be the product of a lot of hard work - in picking 2 suitable servers that work together, and choosing suitable arguments or parameters for each server. So it is suggested that the client can present the results of this work for others to use at some URL. In this case, no new server is created, but rather a "link" to 2 existing servers with particular arguments is set up. And at that link, the client may promote it, explain why it works, etc. User-friendly software will make it as easy as possible for the non-technical to both experiment with different combinations, and link to the result.

    2. At a more advanced level, even a non-technical client may be able to construct a Society. For instance a client may select a combination of remote Mind servers, a remote Action Selection server to resolve the (inevitable) competition between these multiple minds, and finally select a remote Body/World server to run this Society in. To be precise, what the client does is: **Pick a Mind_{AS} server, pass it a list of Mind servers to adjudicate, and then simply pick a World to run the Mind_{AS} server in**. The particular *combination* of Mind servers chosen may be the product of a lot of hard work (in searching for good combinations). So again it is suggested that the client can present the results of this work for others to use at some URL. Again, no new server is created, but rather a "link" to 2 existing servers with particular arguments is set up. User-friendly software will make it as easy as possible for

the non-technical to both experiment with different combinations, and link to the result.

This is like constructing a new Mind server: When a client constructs a new combination of Mind servers like this, it looks very much as if a new Mind server has been created, with totally new behaviour. But in fact it has been done by supplying *new parameters* to an *existing* Mind_{AS} server. Admittedly some of these parameters (the addresses of other Mind servers) may not have *existed* when the Mind_{AS} server was written, and the choice of the combination may be an extremely creative act.

In the above, the client does not *necessarily* need to know anything about how the servers work, or even anything much about AI. The client just observes that certain combinations work very well, and others don't. The role that large numbers of clients acting over long periods may play in experimentation and artificial selection may be very important, as will be discussed in detail later.

2. A technically-proficient **server author** - again any user on the Internet, if they have the technical ability (and also access to a machine to host their new server). They will need to understand how to construct a server, but their understanding of AI does not *necessarily* have to be profound.

   As we have seen above, just specifying an existing server with new parameters can be very much like creating a totally new Mind server with new behaviour. The server author, however, will be writing an *actually* new server. For example:

   1. A technically-proficient server author could write a *wrapper* around an existing, working Mind server, i.e. **Write a new Mind_{M} server**. The most simple form of wrapper would not provide any actions itself, but would just selectively call other servers. For instance, the server author observes that one Mind server tends to perform better in one area of the input space, and another server performs better in a different area. The server author could then experiment with writing a wrapper of the form: *"If input is a particular `x` then do whatever the Mind server M1 does - otherwise do whatever the Mind server M2 does."* The author needs little to no understanding of how either server works, yet still might be able to create a Mind_{M} server that is better than either Mind server itself. For a discussion of such "Nested" systems see [PhD, §18].

   2. An AI-proficient server author might try writing a Mind_{M} server that attempts to provide some actions itself. For example, a server that only *modifies* the existing behaviour in some area of the inputs, such as: *"If input is a particular `x` then take this action `a` - otherwise do whatever the old server does."* The author may need little understanding of how the existing Mind server works. If overriding it in one area of the input space doesn't work (doesn't perform better) he may try overriding it in a different area.

   3. At the most advanced level, AI researchers would write their own servers from scratch. But it is envisaged that even AI researchers will find it useful (in fact, probably *essential*) to write limited wrappers around other people's servers whose insides they don't fully understand (or want to understand).

That is the basic WWM idea. How this scheme will work, and what does it mean, will now be discussed in detail.

2.4 Using other people's agent worlds

First, we note that, among many other things, the scheme is trying to define a protocol whereby researchers can use each other's agent worlds. *Not* by installing the world at your own site, but by leaving the world running on a server at its author's site, and using it remotely. One issue we will have to solve is how does the client see what is going on in the remote world where he is running his selected mind combination?

2.4.1 No user interface

Note the client may not *wish* to see what is happening - he may only want a report back at the end as to how well his agent did - for example, if he is running an automated evolutionary search of different combinations. For example, the client, instead of *watching* his robot picking up cans, may just want to know *how many* cans it picked up over the course of a run. In fact, in this case the client may be able to *calculate* how many cans his robot picked up by examining the state of the world after every action, and keeping his own running total as he goes along. In which case no *extra* report back from the World server would be required. For an example of an automated search with no user interface see [PhD, §4.1.1].

Perhaps the World server provides a URL where the client can see in real time what is going on, *if* he wants to. If this URL is not connected to, no user interface is displayed.

2.5 Using other people's agent minds

The very definition of this system may seem strange to the reader. We are taking for granted that simply being able to match a single Mind server with a World server would be of limited use. Instead we are building a system where clients can put *multiple* minds in the same body. The problem with these multiple minds is that none of them are aware of the others' existence, and each is designed to control the body on its own. Instead of allowing them do that, we simply take their wishes *into account* when running the Action Selection server. For an introduction to these kind of models see [Humphrys, 1997]. The *Adaptive Behavior* audience will be long familiar with these kind of multiple-mind models. Other readers may find them very wasteful and unusual.

Yet, we argue, if 3rd parties are to construct societies of mind using others' components, it will be *impossible* to prevent massive overlap in function and conflict between these components. The realistic approach is to accept overlap as *inevitable* and work on conflict-resolution methods (the AS servers). The AS server will try to allow the expression at different times of most or all of the goals pursued by each mind. With some totally-conflicting goals this may not be possible, so the human clients will finish the job, by looking for specific *combinations* of minds whose conflicts, when resolved by the AS algorithm, results in a useful overall multi-goal creature.

The alternative to a multiple-conflicting-minds model would be to get the components to agree on their function in advance, which is totally impractical.

This is not to say that a group of Mind servers *cannot* agree to divide up function in advance, and the agent mind can consist of a single headquarters Mind_{M} server that knows about this division, and that calls each Mind server, carefully switching control from Mind to Mind. This is possible, but may be very *rare* if Societies are to be constructed by large numbers of researchers. The model cannot assume *in general* that minds make any allowance for the existence of other minds. Users will construct combinations of new minds and old minds, where the new minds did not even *exist* when the old minds were being written. For instance, even with a *perfect* division of function like the collection we just described, no sooner will it be put online than users will start constructing societies involving it and some *new* Minds. And then we are back to an Action Selection problem again.

In fact, as has been discussed elsewhere - see [PhD, §18.3], and also the "brittleness" criticism of classic AI [Holland, 1986] - wastefulness and multiple unexpressed ideas is generally a sign of *intelligence* rather than the opposite. When it comes to intelligence, *parsimony* may be the enemy, rather than wastefulness.

3 Further issues on agent minds

3.1 Mind_{AS} server queries the Mind servers (not Client)

One question then is how complex does the top-level client algorithm have to be? How many servers does it have to talk to? For instance, when there is a collection of Mind servers and an AS server, the client could talk to the Mind servers itself, gather the results and repeatedly present them to the AS server for resolution.

However, when we consider the vast number of possible algorithms for resolving competition, some involving *multiple* queries of each Mind server with different suggestions, it seems more logical for the client to pass the *list* of Mind servers to the AS server at startup, and then let the AS server query them itself, i.e. let the complexity of the competition-resolution algorithm be buried in the AS server rather than in the client. In our model, *the client manages a single Mind server and a single World server*. Which leads to the next question:

3.2 Client talks to the World (not Mind server)

Perhaps the client should pass the World server address as an argument to the top-level Mind server, and let the Mind talk to the World directly.

One reason we do not do this is that - unlike the situation with the AS server talking to the Mind servers - the interaction between the Mind server and World server is not bounded - the run of this Mind in this World may go on indefinitely. It seems better to have this logic in the client rather than in the server. The servers respond to short queries, and the client is responsible for *how many* such queries there are, and what the overall control algorithm is, e.g. implementing time-outs and *repeated* queries if servers do not respond. Or imagine a client where a user is watching the user interface of an infinite run, and deciding, by clicking a button, when to issue the "End run" command.

Another reason we may prefer to define the server queries, and then let a separate algorithm control how many queries take place, is that we may like two Minds to communicate with each other in a conversation, in which case *each Mind serves as the World for the other*. Instead of redefining the Mind server so it can respond to questions such as "Get state", the client algorithm manages this, querying the Mind server for an action, and then sending that action as the state for another Mind server. Imagine two chat programs connected together. Or a "tit-for-tat" online competition [Axelrod and Hamilton, 1981, Axelrod, 1984].

3.3 Low-bandwidth communication

Crucial to the whole scheme of using minds from diverse sources is that we do not impose any restrictions on the type of Mind servers that can be written. Minds can be written in any language and according to any methodology. Minds do not have to explain themselves or how they work. Minds are hidden binaries on remote servers. Minds *cannot* know about each other's goals or insides. Or, to be precise, some minds *may* know how other Mind servers work (and act accordingly), but we cannot in general demand this.

If Mind servers do not understand each other, then they can only communicate by *low-bandwidth* attempts. That is, there is a limit to how much information they can usefully communicate to each other or to the AS server. If, as has already been argued, it will be impossible to prevent large-scale conflicts between Minds, it is the AS server that has to resolve the competition between these strangers who speak no common language. The central question is: **What information does the AS server need from the Mind servers to resolve the competition?** For example, if it just gets a simple list of their suggested actions: a_i it seems it could do little more than just pick the most popular one (if any appears twice). If none appears twice, it seems it could only pick a random one. Perhaps the AS server *rotates* around the Minds, allowing each free reign for a time in rotation. Any such *time-based* Action Selection scheme will be very crude [PhD, §15.1].

3.4 Numeric communication - Q-values and W-values

For any more sophisticated Action Selection than the above, it seems that the Mind server needs to provide *more information*. We will first consider schemes where the servers pass simple numeric quantities around to resolve the competition, yet still do not have to understand each other's goals.

For example, Mind server i may tell the AS server what action a_i it wants to take, plus a weight W_i expressing how important it is for them to win the competition on this timestep. This can be seen as a "payment" in a common currency to try to win the Action Selection (see the "Economy of Mind" ideas of [Baum, 1996]). Or perhaps the AS server, in a bid to resolve conflict, could suggest to all the Mind servers a compromise action a_k and each Mind server could reply with another weight W_i illustrating how much they would dislike taking that action (assuming it is not the action they were originally suggesting). We may define the following weights:

1. The "Q-value" defines how good this action is in pursuit of the Mind server's goal, i.e. expected reward or benefit from this action. Mind server i might build up a table $Q_i(x,a)$ showing the expected value for each action in each state.

2. The "W-value" defines how *bad* it would be for this Mind server to lose the competition on this timestep, and have another action taken. This rather depends on what action will be taken if it loses. Mind server i may maintain a table $W_i(x)$ defining how bad it is to lose (or how much it will "pay" to win) in each state. Or it may judge the badness of a *specific* action a by the quantity: $Q_i(x,a_i) - Q_i(x,a)$.

The usage of Q and W comes from [Humphrys, 1997]. High Q does not imply high W. Q could be high and yet W = 0. For the differences between Q and W see [PhD, §5.5, §6.1, §16.2]. Given a set of Q-values, there are many possible schemes for deriving W-values (discussed throughout [Humphrys, 1997]). The WWM server queries defined in this paper will allow *all* of these numeric schemes to be implemented.

Higher-bandwidth communications than this would seem difficult if we are not to impose some structure on what is inside each Mind server. Hence I will begin the WWM implementation with a *sub-symbolic, numeric Society of Mind*, rather than a symbolic one.

3.5 The role of Mind$_M$ servers

Competition resolution, however, does not *all* have to be done by AS servers looking at Q-values and W-values. Much of the work of combining multiple minds will also be done by hand-coded Mind$_M$ servers, which state explicitly who wins in which state:

> *"As long as the input state is in the region X do what Mind server no. 4006 wants to do,*
> *otherwise, if the input state is in the region Y do what Mind 33000 wants to do,*
> *otherwise (for all other states) give me a "strong" Mind 8000 (i.e. it has lots of currency to spend) and a "weak" Mind 11005, and let them compete under AS server 300."*

In this case the knowledge that Mind server no. 4006 should always "win" the competition when the input state is in the region X is something that the server author has hard-coded. The Mind server did not have to convince the other competing minds (or the AS server) of this fact. In general, a Mind server can implement any general-purpose algorithm that interprets the incoming state, and can call another Mind server at any point in the algorithm. For a discussion of such "Nested" systems see [PhD, §18]. One issue in nested servers will be the possibility of a *circular* call, leading to an infinite loop. It is not proposed to build anything into the protocol to prevent this. It is up to the server authors to prevent. [PhD, §18.1] discusses infinite loops in a society of mind.

To simplify, the above algorithm would be written as a Mind$_M$ server whose "New run" command would be:

```
send "New run" command to M1
send "New run" command to M2
send "New run" command to M3 with arguments (strong M4, weak M5)
```

where the server `M3` is a Mind$_{AS}$ server that can be passed its list of servers as arguments at startup. Then the Mind$_M$ server's "Get action" command (with argument `x`) would be:

```
     if x in region X1 send "Get action" command to M1 with argument x
else if x in region X2 send "Get action" command to M2 with argument x
                  else send "Get action" command to M3 with argument x
```

M3's "Get action" command sends a "Get action" command with argument `x` to *both* M4 and M5, and then makes a decision based on what they return, perhaps querying them further before deciding.

3.6 What is the definition of state and action?

We have so far avoided the question of what is the exact data structure that is being passed back and forth as the state or action. It seems that this definition will be different in different domains. This scheme proposes that we allow different definitions to co-exist. Each server will explain the format of the state and action they generate and expect (most of the time this will just involve linking to another server's definition).

4 Further issues on agent worlds

4.1 Why not separate World and Body servers?

The above model simplified things by having a joint World-Body server. Why not separate further into World servers and Body servers?

The fact is that a World server *is* a Body server. The world only "exists" in so far as it is perceived by the senses. Imagine if we did split the model into World servers and Body servers. The World server would respond to requests for the state of the world with an output `x` and would be sent inputs `a` for execution. The Body server - would do the same thing. It would respond to requests for the state of the world *as perceived by the senses* with an output `x` and would be sent inputs `a` for execution. From the point of view of the client, the two servers are the same type of thing. It's a matter of style which they advertise themselves as.

But there is still a problem with this model of joint Mind-Body servers. It *seems* to indicate that only the World author can write Bodies - which goes against the philosophy of the WWM, where 3rd parties can write add-ons without permission. The question is: **How do we write new Bodies for the same World (if we are not the World author)?**

The answer is that it depends what we mean by writing "new Bodies" for the same World. There is a limit to what we can do with someone else's World without writing our own. The world may have a fixed number of actors - say, one (e.g. a robotic world) - and what we are meant to do is provide the mind for the single actor. We cannot *will* another actor into existence, so all we can do is give the single actor a different type of *Body* than the Body that the World server has given it by default.

And even then, we cannot give it senses that the World has not already given it. All we can do is write Bodies that sense a *sub-space* of the original state `x` provided by the World.

4.1.1 Changing the Body for the World

So when inventing new Bodies, we write them as *new* World-Body servers, which are *wrappers* around the old World-Body server. We will call such a World server a **World$_W$ server** to reflect the fact that it communicates with another server (which may itself communicate in a *chain* of World servers) unknown to the client.

The client only sees the senses as presented by the wrapper World$_W$ server, and sends its actions to the World$_W$ server too. The World$_W$ server talks to the "real" World server. Presumably to it it seems like just another client, requesting `x` and reporting `a`. And to the client, the World$_W$ server looks like just another World server. The senses that the World$_W$ server presents to the client are strictly a *subset* of the raw senses presented by the original World server.

Working with only a sub-space of the original state `x` may sound restrictive, but remember that in general, a "World-Body" server just provides a stream of output `x` when queried about the state of the world. We have not really defined what this output should be, and in fact people may write World servers whose outputs `x` are

deliberately *designed* not to be used as the raw senses of any agent, but rather are designed to be filtered by many different species of 3rd-party Body, in wrapper World_W servers. Note that even having a long chain of World_W servers does not break our basic model that *the client deals with a single top-level Mind server and a single top-level World server*.

4.1.2 Multiple Bodies in the same World

Having accepted that a 3rd party cannot necessarily control the number of actors in a World, how would the World server *itself* deal with multiple actors?

- **Robotic worlds:** For example, say we have a robotic testbed with 2 robots equipped with different senses.

  This would *not* require a World server and multiple Body servers. The senses returned by each are quite different, so this demands different World servers. The logical implementation would be as 2 different World-Body servers, both located at the original site. They will detect each other's existence in the sense that the other robot will appear in the state of the world x as detected through their senses.

  Below we will discuss the special issues with robotics, where each World-Body server can only have one client in control of it at a time. Here with the 2 robots, if a client wanted to do experiments in multi-agent cooperation, they would need to gain exclusive control of *both* World-Body servers at the same time.

- **Virtual worlds (own copy):** A *virtual* World server could create a separate *instance* of the virtual world for each client, so that each client acted alone as a single actor in the world.

- **Virtual worlds (shared):** On the other hand, the virtual World server might provide the ability to *create* a new actor for every client that was using it, all such actors operating in the same world, and detecting each other's existence through state x.

  But again this would *not* require a World server and multiple Body servers. It can be done with a single World-Body server, with multiple clients connected to it at the same time, each sharing an instance of the world with the other clients, rather than getting its own instance. The state x that the server returns is customised for each client.

4.1.3 The joint World-Body model is no restriction

In conclusion, our joint World-Body model is no (or at least, little) restriction for a 3rd party:

1. **New Bodies:** We can change the Body for the World (within limits) by writing a wrapper World_W server.
2. **Heterogenous Bodies:** We can write many different versions (within limits) of these World_W wrapper servers, thus creating many different possible Bodies for the same World.
3. **Multiple Bodies in Same World:** We can add many Bodies to the same *instance* of the World, if the World permits it, by using multiple clients.
4. **Multiple Heterogenous Bodies in Same World:** And since our clients can connect each to different wrapper World_W servers, we can add many Bodies of different types (within limits) to the World.

4.2 What if the Mind cannot make sense of the World?

If we are to allow 3rd parties place Minds in any World-Body they like, and run the result, we must accept that the Mind server may not be able to make sense of the chosen World. For instance the format of the incoming state x may be different to the format the Mind server was expecting. In fact, far from being a special case, this will probably be true of 99 percent of all combinations that a client could choose.

The basic scheme is that we allow this. The 3rd parties need to be given total freedom in their experiments in artificial selection. Combinations that don't work are not a problem. The 3rd parties will only of course advertise the combinations that work. World authors will explain the structure of the state x that they generate, and Mind authors will document the structure of the x they expect, and further servers and clients will act accordingly.

4.3 Real robots

As mentioned, this model could even be used to control real physical robots or other real hardware. There are already a number of robots that can be controlled over the Internet. For an introductory list see the [Yahoo list of robots online]. "Internet tele-robotics" raises some special issues:

1. We may want a scheme where only one client can control the robot at a time. Whereas with a software-only world one can always allow multiple clients (e.g. by creating a new *instance* of the world and body for each client). If the robot allows only one client at a time, we need WWM server commands to *start* a run (block all other clients) and *stop* a run (mark free for another client).

2. The robot owner may want to restrict *who* is able to run a mind on his machine, since some control programs may cause damage. Methods of *security* and *payment* may therefore need to be integrated.

3. A virtual world server requires little or no maintenance. The author can put the virtual world up on the server and then forget about it. It runs forever, servicing clients, even perhaps after the author has left the site.

   A robotic world server, however, demands much more of a commitment. You need to maintain the hardware, recharge batteries, supervise use of it, repair damaged parts of the robot or things in the world, tidy up objects in the world that have been scattered or put out of place, perhaps only allow client access when someone is watching, and so on. It is a lot more of a commitment. You need to fund it and set aside a room for it. As a result most of the Internet-controlled robots so far have run for a limited time only. Longer projects will presumably be possible with *payment*.

These issues have already been encountered in the first experimental Internet robots. For a discussion of the issues see [Taylor and Dalton, 1997]. For example, [Stein, 1998] allows remote control of the robot until the client gives it up, or until a timeout has passed (for clients that never disconnect). [Paulos and Canny, 1996] operate in a special type of problem space where each action represents the completion of an entire goal, and so actions of different clients can be interleaved.

Ken Goldberg and colleagues have operated a number of Internet telerobotics projects. In the first "tele-excavation" project, clients queued for access to the robot. In the robotic tele-garden [Goldberg et al., 1996] users could submit discrete requests at any time, which were executed later by the robot according to its own scheduling algorithm. [Simmons et al., 1997] do something similar. The robotic Ouija board [Goldberg et al., 2000] is a special type of problem where the actions of multiple clients can be *aggregated*. We will define the WWM server commands in this paper with a view to being able to implement *all* of these systems.

4.4 Time

The nature of *time* in this system is interesting. The agent senses a state of the world `x` and then talks to perhaps a large number of servers in order to find an action `a` to execute. Some of these servers may be down. Others may take a long time to respond. The question is: **What happens to the world while the agent is waiting to act?** It seems there are 2 possibilities:

1. **Synchronous World** - The world waits for the agent's next action before it makes the transition to the next state.
2. **Asynchronous World** - The world changes state according to its own timetable, independent of the agent. If the agent wants to behave adaptively, it must sample the state of the world often enough. In this case, what the world server provides is a window onto an *independently-existing*, changing world.

We consider which of these would be used in each type of world:

- **Robotic worlds - Asynchronous:** In a robotic world, if the mind servers do not respond quickly, then the state of the world may have changed completely in the meantime. Perhaps we need some *time-out* system. If the World server receives no answer within the time-out, then the old state of the world becomes "invalid" and it sends a new state `x`.

On the other hand, we could say it is not the World server's problem. We could say it's up to the client to implement time-outs (and re-check the state x) if it wants to behave adaptively. The advantage of this would be to maintain a simple *client-driven* model. *All* servers - the World server, Mind servers and AS servers - could ignore time. These servers would simply respond to requests emanating from clients. The original client controls time and so can set up a system of time-outs.

In a multi-level system, there may be communication going on that the client does not know about. For example, the "Mind" server that the client is waiting for may actually be a **Mind$_{AS}$ server** resolving competition between multiple Mind servers. In this case, the Mind$_{AS}$ server might implement its own time-out. If it hasn't got a reply from all of the Mind servers, it just makes a decision based on whatever suggested actions have come in so far. Alternatively, the Mind$_{AS}$ server may simply wait until it has had a response from everyone, and if that takes too long then the client that is waiting for the Mind$_{AS}$ server will time-out.

It seems that a **Mind$_{M}$ server** *cannot* implement a time-out - it simply has to wait for the Mind server it is calling to return.

- **Virtual worlds (own copy) - Synchronous:** In a virtual world where the client has its own instance of the world, then the World server can simply wait for the client to return an action before it moves on to the next state of the world. Even if the server is down or not responding, that need not necessarily end the "run". We could set it so that the run only ends by agreement, not by mere server failure, which merely postpones the next step.

  In a purely virtual world, the agent could live in a strange, "arrested-time" world. It interacts with the world, the world server is down, it waits days for the next "timestep" to see the next state of the world. Similarly, the world waits days to get the agent's action so it can advance the clock to the next state. Waiting is not a problem for either. A simulated world can run at any speed. Time speeds up and slows down, depending on the servers.

- **Virtual worlds (shared) - probably Asynchronous:** In a virtual world shared with other clients, the world may either move on to the next state after a time-out, implementing whatever actions have come in so far, or else wait for *all* clients to return before moving on. The problem with the latter is if a single client crashes or goes offline, then *all* clients are frozen. So the shared virtual world will probably be Asynchronous, and it is up to the client to sample the state of the world often enough if it wants to be adaptive.

4.5 The name "The World-Wide-Mind"

The name "The World-Wide-Mind" makes a number of important points:

1. **The mind stays at the server:** The name highlights the fact that the mind is not copied but rather stays at the server. We believe that the Web, by allowing documents *remain* at the remote server, and accessing them remotely, provides an outstanding example of *reuse* of data that is applicable to reuse of software as well. Under the "Web-like" model of software reuse, instead of the complexity of installing your own copy, upgrading version 4.0 run-time libraries to version 5.0 libraries, and so on, you link to a remote service. In a Society of Mind constructed according to this principle, the mind will be *literally* decentralised across the world, with parts of the mind at different remote servers. Hence the name.

2. **Parts of the mind are separate from each other:** The name also highlights that the important thing is not the separation of *mind from world*, but the separation of different parts of the mind *from each other*, so that, for example, they can be written and maintained by different authors.

3. **This is separate from the Web:** The name also indicates that this is a different thing to the *World-Wide-Web*. During the recent rise of the Internet, many people have talked about seeing some sort of "global intelligence" emerge. For a survey see [Brooks, 2000]. But these writers are in fact talking about the intelligence being embodied in the *humans* using the network, plus the pages they create [Numao, 2000], or at most perhaps the intelligence being embodied implicitly in the hyperlinks from page to page

[Heylighen, 1997, Goertzel, 1996]. Claims that the network *itself* might be intelligent are at best vague and unconvincing analogies between the network and the brain [Russell, 2000]. Indeed the whole idea of "global intelligence" has had a bad press (rightly, I believe) since the days of Teilhard de Chardin. It is simply incorrect that a large number of individuals in communication will simply "emerge" as a mind. *Not every society is a mind.* Minds are highly structured things, and need to be deliberately constructed, as this paper will attempt to do.

For a *real* society of mind or network mind, we need a network of *AI programs* rather than a network of pages and links. We may actually be able to *implement* this on the existing network of Web servers, running over HTTP, but it must be *designed* as such. It will not simply "emerge" from schemes for linking data on the Web.

4. **This may not even *interact* with the Web:** By separating this from the Web, the name also separates this from existing work that might go under the name of "AI on the Web", namely, AI systems learning from the Web. There are many such systems, the most impressive perhaps being the citation indices CiteSeer [Lawrence et al., 1999] and Cora [McCallum et al., 2000]. The "global intelligence" researchers who have concrete models as opposed to just metaphors [Heylighen, 1997, Goertzel, 1996] have also started by looking at the existing Web.

   But a WWM system is not *necessarily* interested in learning from or interacting with the current Web or its users. We are embedding the WWM in the network not so much because of the prior existence of the Web (though we may make use of that), but mainly because of the future potential of *other WWM servers*.

5 How the WWM will be used in AI

We envisage of course that progress in AI will continue to be driven mainly by professional researchers in AI laboratories. But if these researchers write their AI minds and worlds as *servers*, the AI project could be massively decentralised across different AI labs. As well as allowing AI labs to share work, and specialise, there would also be *a world-wide experiment always on*, with constant re-combination and testing (in public) of everything that has been put online so far.

5.1 Dividing up the work in AI

First of all, AI researchers can more easily specialise. Minds are separate from Worlds, so that researchers can reuse each other's Worlds. Anyone doing, say, AI experiments in learning, can, by writing to this protocol, use as a test bed someone else's World server, and does not have to write his own. He can concentrate purely on devising new learning algorithms. Re-using other people's Worlds [Bryson et al., 2000] will probably be the most common use initially. Learning how to reuse other people's *Minds* may take time.

5.2 Making AI Science - 3rd party experimentation

As discussed above, having to invent your own test environment to test your new algorithm is not only a lot of extra work - it also makes it harder to objectively evaluate the new algorithm. Even if you take care to do a run of the old algorithms in your new test world, the test world might (however unintentionally) be designed to illustrate the good points of the new algorithm. Clearly, being able to compare different algorithms on the *same* pre-built World server goes a long way towards helping AI run objective comparisons of algorithms and models of mind.

But the WWM goes much further than that. By its emphasis on *3rd party* experimentation, algorithms will be subjected to constant examination by populations of testers with no vested interest in any outcome. 3rd parties will ensure that the results can be *repeated* again and again. They will compare many more different combinations of servers under control conditions. (Currently we are still assuming that 3rd parties will be *other AI researchers*. Whether the *general public* will carry out useful tests will be discussed shortly.)

The whole question of how to *prove* one autonomous agent architecture is better than another has become an important issue recently. [Bryson, 2000] points out that, essentially, no one uses each other's architectures: *"There have been complaints .. about the over-generation of architectures"* and (among behavior-based models): *"no single architecture is used by even 10 percent of .. researchers."* Most architectures have performance statistics to support them, but these statistics have had little success in convincing rival researchers to abandon their own favourite models. In particular, if the tester *invented* the test world to show off his new algorithm, there are simply too many uncontrolled variables for the test results to be totally conclusive. Test results can only become conclusive when there is repeated objective 3rd party evaluation. Currently 3rd parties have to go to great effort to recreate the test situation, and this rarely happens. For instance, Maes' model [Maes, 1989, Maes, 1989a] waited years for Tyrrell to re-implement it in a performance comparison [Tyrrell, 1993]. Tyrrell does not get impressive performance for it, but reading his thesis one might argue there are still uncontrolled variables in Tyrrell's implementation. Re-implementing Tyrrell *itself* is a difficult job [Bryson, 2000a] which rather adds to the lack of conclusion about his results.

In any branch of AI, the existence of objective tests that cannot be argued with tends to provide a major impetus to research. This has been one of the main reasons for the popularity of rule-based games in AI [Pell, 1993]. *Robotic soccer* has also taken off in recent years for this reason. Noda's soccer server [Noda et al., 1998] is probably the closest in spirit to the WWM, though the user must still download and install the agent world.

On the WWM, mind and world must be presented *publicly*, and 3rd parties will drive objective evolution of the best solutions. They will test servers in environments their authors never thought of, and combine them with other servers that their authors did not write. The experiments will be outside of the server authors' control. They will do this in public and, if properly written up, this will implement an ongoing objective programme of **artificial selection** (i.e. selection by hand) and **"natural selection"** (machine-automated selection) of agent solutions.

5.3 Artificial Selection

I imagine that many will see the ability of any 3rd party on the Internet to choose and run their own combination of servers as a "cute" but unessential feature. That it is some essentially *patronising* scheme of allowing the public think they are helping with science.

But this misses a couple of points. First, that the only way of allowing *any* professional AI researcher to experiment with servers without permission is to allow *every user on the Internet* to experiment with the servers.

Secondly, in fact, I believe it will be *essential* to the success of the idea that more than just AI researchers can work this system. AI researchers are focused on specific projects, with deadlines and many responsibilities. There are not many of them, and they don't have much time. With 3rd parties, there are millions of them, and they have a vast amount of free time. Many schemes have harnessed the power of the millions of idle and curious Net users, e.g. "metacomputing" projects such as large-scale cryptography cracks or SETI data analysis. Within AI, there have been some evolutionary experiments attempting to recruit large numbers of users. See the [Yahoo list of ALife programs online]. Perhaps the most impressive is the "Talking Heads" language evolution project [Steels and Kaplan, 1999] in which 4000 agents have been constructed and tested by online users [Steels, 2000]. Such ideas are in fact fundamental to the Internet. Even the idea of *linking* itself in the Web is a classic example of harnessing the power of large numbers of people. Other people do some of the work for you, by tracking down sites and presenting you with a pre-built list of links.

Here with the WWM, the idea of a client presenting at a URL a pre-built combination of servers is deliberately modelled on the Web idea of a pre-built selection of links. Working the system will not be as easy as making a link on the Web. It will require some ability and interest - though perhaps no more so than the interest ordinary people have in raising animals and infants. People will write user-friendly software to make it easy to be a non-technical client, experimenting with combinations of pre-existing servers, taking part in ongoing competitions to beat the highest score in a particular World. Obviously the amateurs will report their successes in what will probably be a somewhat *haphazard* fashion. It will be up to the AI professionals to make sense of what they see, and investigate promising leads and write them up in a scientific manner.

But the power of *artificial selection* by large numbers of amateurs should not be underestimated by science. Putting unseen-before minds into unseen-before worlds, years after the research groups that made both minds and worlds have vanished, may be of real benefit to science. 3rd parties will run old minds in new worlds, and new minds in old worlds. They will drop new minds into old collections, and run combinations that make no sense. It is *certain* that they will run combinations of servers that the scientists never thought of.

The professionals may be sceptical of the value of this, but millions of people experimenting with different mind/body/world combinations year after year on the network would represent a richer experimental milieu than anything AI has ever yet built. Thousands of years of artificial selection by farmers and breeders across the world (non-scientists) is now recognised as one of the most thorough scientific experiments in history. Since modern science arose, hardly any *new* animals or plants have been successfully domesticated, which indicates that pre-scientists really did do all the major experiments [Diamond, 1997]. And of course the *scope* of what artificial selection has been able to produce, from fruit and vegetables to dogs or racehorses, is breathtaking, and was one of the central inspirations for Darwin's theory [Darwin, 1859, §1, "Variation under Domestication"]. With AI, large-scale artificial selection projects cannot begin unless the AI work comes online.

5.4 How 3rd party *AI researchers* will use the scheme

To continue our theme in defence of 3rd parties, it is often forgotten that many *AI researchers* are excluded from the AI project as well. The WWM scheme will allow AI researchers in poorly-funded labs, in distant and poor countries, isolated postgraduate students, and so on, to participate in the great AI adventure. Not for charitable reasons, but because their exclusion is a loss to the science.

They will make a more sophisticated use of it than the amateurs. They will write new servers themselves, either from scratch, or by partial reuse of existing servers, by writing Mind_{AS} servers, Mind_{M} servers and World_{W} servers. They will re-use others' work in controlled experiments. They could take an existing world, body, problem, and basic collection of minds, and just work on simply adding one more mind to the seething collection. Things like:

1. 1st party makes World.
2. 2nd party makes Mind for World.
3. 3rd party makes Mind_{M} which in state `x` does something, otherwise does what 2nd party Mind does.
4. 4th party makes different Mind for World.
5. 5th party makes Mind_{M} which in state `y` does what 4th party Mind does otherwise does what 3rd party Mind_{M} does.

And so on, with people modifying and cautiously overriding what already works. Of course they are not *actually* modifying what already exists in the sense of changing it for other users and removing the old version. Making a wrapper server simply means that 2 servers now exist instead of one. The wrapper author cannot force anyone to use the new wrapper in preference to the old server.

By setting up a system whereby many authors, acting over different times, will contribute to constructing a Society, the WWM provides AI researchers with the ability to do a lot more than just reuse other people's test worlds. In every field, the Internet has already allowed *marginalised*, distant and poorly-funded researchers participate in international research like never before, from access to primary literature that may not exist in *any* library in the client's *country*, to access to remote databases and software libraries. The WWM is simply continuing this trend.

5.5 Bring every agent online

Part of this proposal is a plea for recognition of the untapped potential in AI - the vast number of minds and worlds that are offline. Some of this comes from my own experience with putting an agent mind online. For I was one of the first people to put an AI mind on the network, an "Eliza"-type chat program in 1989 [Humphrys, 1989].

The original "Eliza" was introduced in [Weizenbaum, 1966]. Now I must say upfront that Eliza-type programs have little to do with serious AI - their understanding of conversation is an illusion. But this does not affect the argument. They are *behaviour-producing* systems, like our postulated Mind servers. My point is that many people conversed with the original Eliza (not online of course, for there was no network), but *it did not stay accessible*. Soon, the experiment was over, written up, and the original version of Eliza remained largely inactive until the modern era. In 1989 I put my own Eliza-type program, "MGonz", online on BITNET. Many people talked to it, but soon (in fact, by 1990), MGonz had ceased to interact with the world.

A *brief*, finite interaction with the world, seen by only a few people, and normally not even online, is the *norm* in autonomous agents research. In this field it has become acceptable not to have direct access to many of the major systems under discussion. How many action selection researchers have ever *seen* Tyrrell's world running, for example? [Tyrrell, 1993] How many robotics researchers have ever *seen* Cog move (not in a movie)? [Brooks et al., 1998] Due to incompatibilities of software and expense of hardware, we accept that we will never see many of these things ourselves, but only read papers on them, and sometimes watch movies. This situation seems normal, but if we ever *get used* to direct access to remote agent minds and worlds, it may come to seem like bad science not to allow it, and to only report offline experiments that were seen only by the creator of the agent.

But it is not just watching the agents that this is aimed at, it is *interacting* with them. The problem with inactive agents is that the only experiments run with them were the ones their creator thought of doing. But as we have argued, AI researchers are often too limited in time and resources to explore fully the possibilities of their creations. It is as if animal species only got to live through one individual and one lifespan. As AI develops, we should begin to regard the inactivity of our growing list of old creations as a *loss*, like the silence of extinct species. The WWM aims to put an end to this inactivity.

The invention of CGI and other technologies has recently resurrected some of the old agent minds of AI, including the Eliza-type programs [see Yahoo list of AI programs online]. The WWM will vastly accelerate this process, by bringing many of the recent autonomous agents minds online in re-usable forms where they can be driven by remote programs. We aim to take **all of the Minds and Worlds that human ingenuity can create**, and get them all **online and interacting with the world indefinitely**. To get AI to move away from isolated experiments, and instead develop its own rich, world-wide, always-on, ecosystem to parallel the natural one.

6 Objections to the model

It is important to ask, if this scheme is going to be so useful, why AI has not taken this direction in the past. The following may be reasons why (or possible objections):

1. **Co-operating is too much trouble. -** In the past researchers have not seen the benefits of dividing up the work. As discussed above, many researchers still have the impossible dream of doing everything themselves, such as the CYC project [cyc.com]. Two similar projects, GAC [mindpixel.com] and Open Mind Commonsense [commonsense.media.mit.edu] are online, but are attempting to use the Net to get people to *teach* the centralised agent mind, rather than having the agent mind distributed on the Net.

   Indeed many presentations in the Animats or evolutionary fields still seem to assume that one lab can do it all. Some of them recognise the immensity of the problem as we scale up, but when faced with the complexity of dividing up the work, defining communication protocols, and coordinating the results, most have retreated back into either designing whole agents (but saying this is alright for simple agents) or else producing specialist components which may or may not ever be used as part of a larger system.

2. **How do we divide up the work? -** Part of the problem, I believe, is in the mental model of *reuse* that is being used. It is imagined perhaps that the components being reused need to be *understood*. That their source code needs to be merged with other source codes. That all binaries need to be installed at one location. That components will engage in high-level reasoning and negotiation with each other (rather than simply be mutually-incomprehending minds). And finally, that components will not overlap - that each will have its own well-defined function.

These are all reasons why re-use is difficult in the software world in general. But in the model of reuse proposed here, none of these things are necessary. It is not necessary to have a clear definition of how the work should be divided up. It is not necessary for components to understand each other. It is not necessary to install anything. Components being re-used can *remain* at the remote server, are used as a service from there, and are not fully under the client's control. Which leads to the next reason:

3. **Researchers do not want to be dependent on other people's work. -** What if the remote server is down? Or the author has made changes to it without telling us? Or removed it permanently?

   Part of the problem, I believe, is models of mind in which the loss of a single server *would* be a serious issue. Instead of models of mind where hundreds of similar servers compete to do the same job, researchers have been assuming the use of *parsimonious* minds where each component does a particular task that is not done by others. Certainly, in the early stages of the WWM, with few servers online, clients may feel that their constructed minds are very fragile and dependent on the servers. But some clients will continue to add more and more "unnecessary" duplicated minds to their societies. In a model of mind with enough duplication, the temporary network failure (or even permanent deletion) of what were *once* seen as key servers may never even be *noticed*.

4. **But *some* servers will be indispensable. -** Yes, this is true. While duplicate models of mind can take us a long way, *some* servers will be indispensable. We could have a Mind_{AS} server that collects suggested actions from many Minds, and if some of them are gone it will run with whatever suggestions are left. But the Mind_{AS} server itself is essential, as is the World server. Like pages that we link to disappearing from the Web [Humphrys, 1999], how can we cope with the disappearance of a server that we *need*?

   The basic answer is that if it is important to us, we will copy it (if it is free) or buy it or rent it. Then we either set up our own server, or continue to use the remote server and just keep our copy offline as backup. Here's how in practice one might be running a large, complex Society of Mind with actually very little risk: Imagine that our top-level AS server is a well-known, standard type that we can get our own copy of. (Not that we actually *use* it. We continue to use a remote server. But we have our own copy just in case.) Say the World server is a popular test world that is implemented at multiple sites (we just use our favourite one). And we use hundreds of remote minds in a complex society. For all of these we will take the risk of some of them vanishing, and see little reason to buy or copy any of them ourselves. After all, other new and interesting Mind servers are coming online all the time.

5. **Models of Broken links and Brain Damage** - Broken hyperlinks are a problem with the Web model of remote data, and the equivalent of broken links will happen with any scheme that uses remote servers.

   As discussed above, one way of making a Society more robust would be to add "unnecessary" duplication. For this to work smoothly, we need an Action Selection scheme where a Society with n identical Mind servers trying to do the same thing, plus other servers, will behave more or less the same as a Society with 1 Mind server trying to do that thing, plus other servers. Then we can add extra copies of the Mind server located at different sites, and won't even notice if some of those sites are down. This property is not true of all AS schemes, though. [Humphrys, 1997] shows that it is true of *individual-driven* AS schemes such as Minimize the Worst Unhappiness, but is not true of the more common *collective* AS schemes.

   So using a model like the above, Societies will degrade *gradually* as the number of broken links increases. In the above work I explicitly addressed the issue of brain damage in a large society of mind [PhD, §17.2.2, §18]. The reader might have wondered what is the point of a model of AI that can survive brain damage. After all, if the AI is damaged, you just fix it or reinstall it surely? Here is the point - a model of AI that can survive *broken links*. This leads to the more general reason why this whole approach to remote re-use has not been used:

6. **Models of duplicated mind are poorly developed.** - We have argued throughout that if the work is to be divided up in AI it will be *impossible* to avoid massive overlap and duplication of function, and resulting conflict. We need models of mind in which a state of conflict is totally expected. Unable to get server authors to agree, we will instead selectively override, censor, and subsume old servers instead of re-writing them (or vainly trying to get the server author to re-write them). Such duplicated models have been argued for [Brooks, 1986, Brooks, 1991, Minsky, 1986] but *parsimony* is still popular. We will also need Action Selection that can resolve competition between minds that barely understand each other [Humphrys, 1997].

In a traditional system where a *single* designer writes the whole system, he can make deliberate *global-level* decisions in the interests of the whole creature, and there is no need for the decision to emerge from local rules. But with mind servers from diverse sources, the need for Action Selection based on local rules re-emerges.

It is clear enough to see how *sub-symbolic* conflict resolution can occur via numeric weights. But if Minds are strangers written by different authors, in what *symbolic* language could they communicate? Which leads to the following objection:

7. **It is premature at symbolic level to attempt to define mind network protocols.** - This is probably true. Since even before the Web, researchers have debated the possibility of standardised symbolic-AI knowledge-sharing protocols, with [Ginsberg, 1991] arguing that it is premature to define such protocols. Recently this debate has continued in the Agents community as the debate over defining standardised *agent communication languages*. See a recent survey of many approaches in [Martin et al., 2000], who then define their *own* approach. Agreement is weak, and it may be that the whole endeavour is still premature. For example, some of Minsky's students [Porter et al.] attempt to implement a Society of Mind on the Internet, but insist on a symbolic model, with which they make limited progress. Indeed, Minsky's work may have had little impact in the sub-symbolic world because of his hostility to that world [Pollack, 1989].

   We argue, though, that it is not totally premature to start defining mind network protocols at the sub-symbolic level. There are already many schemes of numeric weights, and Action Selection based on weights, in the literature. The sub-symbolic WWM in this paper has been designed so that all current numeric agent architectures (that the author is aware of) can be implemented under the scheme.

   There will no doubt be further sub-symbolic (and later, symbolic) protocols. But designing the early ones for simple numeric weights will give us an idea of how to do it in the future for more *heterogenous* agents with *different representations* [Minsky, 1986, Minsky, 1991]. This will be the first in a long *family* of protocols.

8. **"Agents" researchers (or other branches of AI) have already done this.** - No they haven't. Consider how the field of *Distributed AI* has developed. For surveys see [Stone and Veloso, 2000, Nwana, 1996]. DAI has split into two camps:

   1. Distributed Problem Solving (DPS) - where the Minds are cooperating to solve the *same problem* in one Body.

   2. Multi-Agent Systems (MAS) - where the Minds are in different Bodies. We have 1 mind - 1 body actors, and then coordination of multiple actors. This is what the field of *"Agents"* has come to mean. Indeed, [Nwana, 1996, §4.3] makes explicitly clear that our servers are not Agents.

   This is neither of these two, but rather is multiple minds solving multiple problems in one body. If anything, it is closer to the field of **Adaptive Behavior** and its interest in whole, multi-goal creatures whose goals may simply *conflict*. Agents researchers also tend to work at the symbolic level only, rather than the sub-symbolic as we do here (and as many people do in Adaptive Behavior).

   Artificial Life and evolutionary researchers are certainly interested in collective, and even collective network-based models [Ray, 1995], but again the minds are localised, as in the MAS approach. In [Ray, 1995] it is a *society of agents* that is distributed across the network, not a *single agent mind*.

   Machine Learning (e.g. Reinforcement Learning) researchers have tended to focus on solving a dedicated problem, rather than juggling many partially-solved conflicting problems. That it, they tend to take the DPS approach.

9. **Virtual-world researchers have already done this.** - No they haven't. They have tried to establish standard technology for displaying worlds, such as VRML. Here we establish a framework within which state and action data may be sent back and forth, but the *format* of the state is left to be defined by each server. These researchers also concentrate on user-driven avatars, with a graphical UI. This considers AI-driven actors, with possibly no UI at all.

10. **Tele-robotic researchers have already done this.** - No they haven't. They have concentrated on user-driven control from web pages, rather than remote machine-driven control.

11. **The network is not up to this yet.** - Possibly true. *Simple* WWM societies can certainly run on today's network. It may be that a Society with a large number of servers in multiple layers will operate very slowly on today's network. But that will change.

12. **There is a chicken-and-egg problem. -** It is true that until *other* people put up their worlds, minds and AS mechanisms as servers, there is not much attraction in converting to using servers oneself. One thing that may speed the adoption of this scheme, though, is currently there is no easy *alternative* method in many areas. For example, how would one make Tyrrell's agent world [Tyrrell, 1993] easily available to researchers? Going the server route seems almost as easy as any other.

7 Miscellaneous issues

7.1 Hidden server insides

The internal server workings do not have to be made public. The only demands are that the server replies to external requests according to the protocol. The Mind server can be a symbolic mind, a neural network, a genetically evolved program, or anything else. It does not have to tell us. This will be important for commercial servers who want to protect their investment, or *sell* access to their server. It may also be important for academic projects. Although it could be argued that *all* academic projects should publish their source code (and there is no excuse not to, now that the Web exists) so that experiments can be replicated.

An interesting side effect of hidden server internals may be a more level playing field between different types of AI. Currently people *tend* to look for algorithms only within a particular sub-field - neural net researchers look for other types of neural net, symbolic AI researchers look for symbolic routines, and so on. Here, each algorithm will stand on its merits, and it may be *better science* if we do not know, at least initially, what is inside the server. The doctrinaire neural net researcher may be embarrassed to discover that the excellent server he has been using for years is in fact a symbolic AI server. Such objective symbolic v. non-symbolic competition has in fact already occurred in robot soccer [Stone and Veloso, 2000].

It will be interesting to see how this develops. Some researchers may refuse to use servers unless they know how they work. They will argue that it is *bad science*, since any systems built using such servers can only be *replicated* as long as the server owner continues to provide the secret and unknown service. Others may argue the above case - that it is a breath of fresh air (and *good* science) to be able to judge algorithms entirely on *performance* without, at least initially, knowing anything else about them. The answer really is that it will be good science if, while server authors *do* explain in some detail what is inside, such information is routinely *ignored* as different combinations of systems are constructed and tested on merit.

7.2 Credit

Allowing servers to be used as components in larger systems is central to the WWM idea. One issue though is: **Can you *track* how your server is being used?** Imagine that your server is being called by a popular Mind_M server. To the outside world, does the Mind_M server get all the credit? Can a 3rd party look at a successful Society, and see all of the servers involved in it at all levels? Or do they just have to assign all the credit to the top-level server, not knowing what is behind it? We suggest the following:

1. Every server has a URL.
2. At that URL they link to the URLs of every server they are calling.
3. When they call another server, they provide it with their URL as part of the query.
4. So each server, at its URL, can link to all servers it calls *and* all servers that call it.
5. 3rd parties may read these lists, and follow chains of credit through the Society, even if the server authors are not involved.

The latter point is important. On the WWM we expect that thousands of server authors will leave their servers to be run in the research community long after they have gone. Hence we want these lists to be *published* online, rather than simply known internally by the programs or recorded in logfiles.

An interesting question is whether you could write a *malicious* wrapper server, where the Mind_{M} server does not acknowledge that it is calling another Mind server, and tries to take the sole credit for the functionality. There are many possible answers to this (e.g. servers publish their usage logs online), but we doubt it will be an important issue. More likely the AI community will simply *ignore* servers unless they come (i) from serious and respectable sources and (ii) explain in exhaustive detail how they work and what other servers they are calling.

7.3 Learning servers

If Mind servers learn from interacting with a World, while part of a Society, where is that new knowledge stored? One might say it should be stored at the Mind server, and that seems reasonable in many cases (if the clients can tolerate the fact that the server may change). There are, however, a couple of problems:

1. First, the server may learn erroneous things. For instance, a client uses a Mind server, and gets a suggested action `a`. The client reports the new state `y` and the Mind server learns that action `a` led to state `y`. But the client forgot to tell the Mind server that its action was not actually obeyed (it lost the competition), so it was *some other action* that lead to state `y`. The client may even be malicious. AI programs that learn from user input online have found that many users input nonsense or misleading information [Hutchens, undated]. The integrity of the Mind server cannot be compromised by a buggy or malicious client. So we suggest that **the Mind server learns relative to a particular client only**. i.e. It stores a file of knowledge that is only used with that client. What you teach it may only change how it interacts with *you*, not how it interacts with others.

2. Some of the learning may only make sense relative to a particular *Society* (e.g. the W-values), and so again we suggest it learns relative to this client only.

This does not mean that other clients cannot access the new knowledge. It just means they have to do so explicitly: *"Give me the Mind that you learnt with client* `c`*"*.

7.3.1 Learning Temperature

A Mind server that learns also raises the question of which of these we want:

1. A pre-built server that has *already* learnt what it wants to do [pure exploitation].
2. A new version to start learning from scratch, i.e. whose initial actions may be completely random [pure exploration].
3. A server that has already learnt some *preferences*, but still engages in some new exploration [some exploitation, some exploration].

We may pick one of these at the start of the run, or even half-way through the run we may decide to get the server to go back and re-learn. We can control all this by passing a "Temperature" parameter to the server. Temperature = 0 means we want the server to exploit its current knowledge with no exploration. High Temperature means we want the server to do a lot of exploration. As Temperature `-> infinity` the server tends to engage in maximum exploration. The use of the word "Temperature" is explained in [PhD, §2.2.3]. Exactly what counts as a "high" Temperature for this server will be explained by the server at its URL. Rather than just provide the Temperature at the start, we might provide it on a step-by-step basis, so that at any point the client may send Temperature = 0, which means "Send me your best action, based on your learning so far". Generally, the temperature will decline over the course of the learning run. We have two basic strategies:

1. **Server maintains temperature -** The temperature is initialised, either explicitly by the client, or the server is left to pick a reasonable high temperature. The server maintains an internal value for temperature, and decreases it every step so that by the end of the learning run it will have reached the minimum temperature (it stops learning). The client will need to tell the server how long the run will be. The client makes requests of the form `Get action (x)`

2. **Client maintains temperature -** The client maintains the temperature value, and passes it to the server with every request. The client makes requests of the form `Get action (x,temperature)`

7.3.2 Q-Temperature and W-Temperature

If the world changes, we may ask the Mind server to re-learn its Q-values from scratch, i.e. increase the Q-Temperature. If the collection of minds changes (i.e. the *competition* changes), we may ask the Mind server to re-learn its W-values from scratch, i.e. increase the W-Temperature.

Part 3 - Implementation

8 Implementation

Having defined in the first part of this paper how a WWM scheme could work, and what it would be good for, we shall now move towards an actual implementation. First we need to define what the standard mode of operation will be. Will clients connect to servers for long periods of time? Or will they connect, carry out some transaction, and then disconnect? First note the differences with the Web:

- The Web is a short, non-looping process - Start request - Send URL - Get all files - End request.
- CGI is similar - Start request - Send arguments - Get all data returned by program - End request.
- The WWM, however, is a looping, *indefinite-length* process - Start run - Run indefinitely (Get state, get action, execute action, get state).

  An explicit "End run" command might be inserted, perhaps by the client, or by a server when it has detected some condition. Or the loop could run for a pre-determined amount of time.

8.1 Short, limited-length, client-server transactions

One way of breaking this indefinite loop down might be to treat each individual interaction with a server as a short, non-looping process. The server responds to a short query with a response that will return within a limited time. The server does not know when, if ever, it will receive the next query. *Other* algorithms implement loops and more complex logic repeatedly using these primitive server queries.

8.2 Client algorithm

The client software, which is driving a single top-level Mind and World, will implement a program something along these lines:

1. For each server:
   - Connect to server - Start request - Tell server to start a new run for this client - Receive a unique run ID, so that you can identify yourself later - End request
2. Repeat:
   1. Connect to World server - Start request - Send run ID to identify yourself - Query state - Get state `x` - End request
   2. Connect to Mind server - Start request - Send run ID to identify yourself - Send state `x` - Get action `a` - End request
   3. Connect to World server - Start request - Send run ID to identify yourself - Send action `a` - Get new state `y` - End request

4. Connect to Mind server - Start request - Send run ID to identify yourself - Tell it new state `y` - Receive confirmation - End request

3. For each server:
   - Connect to server - Start request - Send run ID to identify yourself - Send "End run" command - Receive confirmation - End request

To clarify, the *non-technical client user* will use the servers through standardised client software. It is the *client software* that will implement this overall control algorithm. The non-technical client user will see none of this.

8.2.1 The server may be involved in many runs

The unique run ID is because the server may be simultaneously involved in many other runs with other clients. The server must keep the details of each run separate from each other. [Paulos and Canny, 1996] use a unique run ID like this in Internet robotics, where a World is servicing multiple Minds. Here also the *Mind* server may be involved in *other Societies* running at the same time. So the World-Wide-Mind is even stranger than having bits of your mind distributed around the world. It means that **bits of your mind are simultaneously running in other minds**.

8.2.2 The client controls time and may implement time-outs

The Mind server does not talk to the World server directly. Rather, the servers respond to short, finite-length requests. The client algorithm controls *how many* such requests occur. It is up to the client how long it runs the above loop for. The client may also implement a time-out. If the Mind takes too long to reply, the client could:

1. Abort the Mind request, query the World again to make sure the state is up to date, then query the Mind again.

   or:

2. Wait for the Mind to reply, so we know it is back online again. Then *ignore* its reply, query the World to get up to date, and then query the Mind again.

Similarly, if the World is down, the client may wait until it is back up, and then requery the Mind with the new state, instead of blindly executing the old, unexecuted action. This scheme could allow for a variable use of time, where the client may take *days* to come back to each server with the next request.

8.2.3 This is not a stimulus-response model

Continuing the discussion about time, it is important to reiterate that the above does *not* define the Mind as a stimulus-response machine. The Mind is simply receiving a periodic update about the state of the World. The Mind may run according to a different clock to the World:

1. If the World changes slowly then a large number of `x` in a row may be the same. In this case the Mind is receiving more updates than it needs, and if the model demands that it return an action in response to each of these updates, then we will want to define as one of our actions an action for "Do nothing".

2. Alernatively, just because the current state `x` is the parameter that is sent along with the "Get action" query, does not mean that the action returned is a function of `x` *alone*. The Mind may suddenly start taking actions even though `x` has not changed. The Mind can be remembering *all* previous states, and making its judgement based on that knowledge. It can be building a world model. It can have internal clocks that cause it to change plans according to *time-based* action selection [Ring, 1992, Blumberg, 1994, Whitehead et al., 1993, McFarland, 1989]. It can be learning, starting with random actions, and changing its policy as it goes along. It can be engaged in long-term or short-term planning. It can be symbolic or non-symbolic.

8.3 Mind$_{AS}$ server algorithm

The Mind$_{AS}$ server responds to queries like any other server. *Inside* its "Get action" query is some complex logic interrogating a list of Mind servers to find a winning action. This may be a loop but, unlike the client, it will be a finite-length loop, not an indefinite-length loop.

Similarly, the Mind$_{AS}$ server will receive an "Inform it about new state y" command after each action is executed. *Inside* this command it will send an "Inform it about new state" command to all of its subsidiary Mind servers, along with extra information that only the Mind$_{AS}$ server knows, such as whether they were obeyed or not.

8.3.1 The Mind$_{AS}$ server may also implement time-outs

The Mind$_{AS}$ server may implement time-outs. It is periodically sending a query to *all* Mind servers. It cannot afford to go round them in rotation. It cannot afford to wait until Mind server M1 has returned before sending a request to M2. Instead it must send requests to all of them in parallel, and receive the replies as they come in. With multiple Mind servers there is a much greater chance of some being slow, offline, or even gone altogether (broken links). Any sensible Mind$_{AS}$ server will implement a time-out. If some of the Mind servers do not respond within the time-out, it will make a decision based on whatever actions *have* come in.

8.4 The servers (and client software) may implement any general-purpose algorithm using the server queries

Clearly the above client algorithm pseudo-code could be made more efficient. Perhaps there is no need to connect to the Mind after each action - it can get to find out what happened (what the new state is) *next time around the loop* when it is asked for a new action. And next time round there is no need to query the World again - we already know what the new state is from when we executed the action. But the point is that it is up to the *client* to write this program. It will not be laid down in the protocol. Similarly, a server may implement any algorithm it likes provided it responds to the set of queries expected of it.

9 List of server queries

The definition of the WWM comes down to this, the definition of the possible queries and responses of the servers. The client software may implement any general-purpose algorithm based on these queries and responses. The Mind servers, the World servers, the Mind$_{AS}$ servers, the Mind$_{M}$ servers and the World$_{W}$ servers may each implement any general-purpose algorithm based on these queries and responses, provided that *they themselves* respond to these queries.

9.1 World server

Request name	Argument data	Return data

New run	1. (OPTIONAL) Client URL (client may be a real client, or another $World_W$ server)
	2. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the World server. These are parameters for the world, e.g.:
	  1. synchronous or asynchronous
	  2. shared or non-shared
	  3. start new world or join existing world
	  4. virtual world size
	  5. parameters defining number and type of other objects in virtual world
		1. Confirm (robot now in use, new copy of virtual world set up for this client, new actor in existing virtual world set up, etc.)
		2. world run ID
		3. (OPTIONAL) world display URL
		or
		1. Refusal (client blocked, client URL not valid, failure in following trail of credit from client URL, payment or authentication required, robot already in use, bad parameters)

Get display URL	1. world run ID	1. (OPTIONAL) world display URL
"No operation" (Possibly used as a periodic clock timer, or just to inform the server that the client is still running.)	1. world run ID	1. Confirm
Get state	1. world run ID	1. x
Execute action	1. world run ID 2. a	1. y 2. (OPTIONAL) Score (points scored by this single action, according to some scoring system at the World server). This score could be used to drive automated searches with no user interface.
Reset (Reset the world as it would be at the start of a run. e.g. We are trying to solve a problem. Previously we were just learning. Now we want to *test* our knowledge.)	1. world run ID	1. Confirm (This may or may not reset the score.)
Reset score	1. world run ID	1. Confirm
Get current score	1. world run ID	1. (OPTIONAL) Score (total points scored so far in this run).

End run	1. world run ID	1. Confirm (Display URL is removed, robot is freed for other use, etc.) 2. (OPTIONAL) Score (total points scored over course of run).

Notes:

1. The unique world run ID is known only to the client, and used to identify itself each time when it returns with a new query.
2. The state and the action are indefinite-length, undefined streams of plain text (terminated by `</data>`) to be interpreted by the servers.
3. The World should probably support an action for "Do nothing".

A World$_W$ server has the same interface as a World server.

9.2 Mind server

Request name	**Argument data**	**Return data**
New run	1. (OPTIONAL) Client URL (client may be a real client, or another Mind$_M$ server) 2. (OPTIONAL) world display URL 3. (OPTIONAL) world run ID 4. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for the mind, e.g.: 1. maximum allowable timeout before mind must return an action	1. Confirm 2. mind run ID 3. (OPTIONAL) mind display URL (Mind may display at some URL information about how it is being used on this run, what it has learnt, etc.) **or** 1. Refusal (client blocked, client URL not valid, failure in following trail of credit from client URL, world display URL not valid, failure in following trail of credit from world display URL, payment or authentication required, bad parameters)
Get display URL	1. mind run ID	1. (OPTIONAL) mind display URL
"No operation"	1. mind run ID	1. Confirm
Ready to suggest action?	1. mind run ID 2. (OPTIONAL) current state x	1. Ready to suggest an action. **or** 1. Cannot suggest an action at this time or in this state (e.g. has terminated, or is waiting for pre-conditions to be met).

Get action	1. mind run ID 2. x	1. a 2. (OPTIONAL) Q (predicted points that will be scored by this action, see below) **or** 1. Cannot suggest action at this time or in this state. Note that the action "Do nothing" is *not* equivalent to "Cannot suggest action".
Inform it about state (Inform it what happened when the action was executed)	1. mind run ID 2. y 3. (OPTIONAL) Score (points scored by action according to World)	1. Confirm 2. (OPTIONAL) Q (points scored by this action, according to the *Mind's* way of scoring points, which might be different to how the World sees it).
Reset score	1. mind run ID	1. Confirm
Get current score	1. mind run ID 2. (OPTIONAL) Score (points scored so far in run according to World).	1. (OPTIONAL) Score (points scored so far in run according to Mind).
End run	1. mind run ID 2. (OPTIONAL) Score (total points scored in run according to World)	1. Confirm 2. (OPTIONAL) Score (total points scored in run according to Mind)

Notes:

1. A Mind server may call other Mind servers, thus setting up its own run with them, and its own run ID. Presumably Minds will start other Minds with progressively smaller "maximum allowable timeout" parameters.

9.2.1 Additional Mind$_L$ queries

A Mind$_L$ server is a Mind server that learns, and supports the following additional queries:

Request name	Argument data	Return data

New run - Mind$_L$ arguments	1. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for the mind, e.g.:
	  1. Numeric values of a set of rewards from which the mind will now start learning
	  2. Q-Temperature
	  3. Proposed length of Q learning run (for use in declining Q-Temperature)
	  4. discounting factor (in Reinforcement Learning)
	Parameters for a neural network mind might include:
	  1. no. of hidden units
	  2. learning rate
	2. (OPTIONAL) Other Client URL. Say the server stores its learnt knowledge relative to each client. This parameter is to say "Give me the version of the Mind you have constructed by learning with the client at this URL."
		1. Confirm
		or
		1. Refusal (bad parameters, other client URL not valid, no data saved relative to other client URL)

Get Q-Temperature	1. mind run ID	1. (OPTIONAL) Q-Temperature
Reset Q-Temperature / World has changed	1. mind run ID 2. (OPTIONAL) Proposed length of next Q learning run	1. Confirm (will reset Q-Temperature to something sensible)
Send explicit Q-Temperature	1. mind run ID 2. Q-Temperature 3. (OPTIONAL) Proposed length of next Q learning run	1. Confirm
Get action	1. mind run ID 2. x 3. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for this step only, e.g.: 1. Q-Temperature	1. a 2. (OPTIONAL) Q (predicted points that will be scored by this action) **or** 1. Cannot suggest action at this time or in this state.

9.2.2 Additional Mind$_i$ queries

A Mind$_i$ server is a Mind server that accepts it may not be the only mind in the body, and supports the following additional queries:

Request name	**Argument data**	**Return data**
New run - Mind$_i$ arguments	1. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for the mind, e.g.: 1. mind "strength" 2. W-Temperature 3. Proposed length of W learning run	1. Confirm **or** 1. Refusal (bad parameters)
Get mind strength	1. mind run ID	1. (OPTIONAL) mind strength
Change mind strength (Could ask for a new instance of the mind with a different strength, or there might be some reason to keep the current instance and change its strength)	1. mind run ID 2. mind strength	1. Confirm
Get W-Temperature	1. mind run ID	1. (OPTIONAL) W-Temperature
Reset W-Temperature / Collection has changed (new competing Minds, or old competing Minds have gone)	1. mind run ID 2. (OPTIONAL) Proposed length of next W learning run	1. Confirm (will reset W-Temperature to something sensible)
Send explicit W-Temperature	1. mind run ID 2. W-Temperature 3. (OPTIONAL) Proposed length of next W learning run	1. Confirm

Get suggested action with values	1. mind run ID
	2. x
	3. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for this step only, e.g.:
	  1. Q-Temperature
	  2. W-Temperature
		1. (OPTIONAL) a (my suggested action)
		2. (OPTIONAL) Mind server URL (of the Mind I want to call now)
		3. Q (a measure of how good this action is - how much this action will benefit me).
		4. W (how much I am prepared to pay to win this competition). i.e. We must have some idea what will happen if we don't win.
		
		or
		
		1. Cannot suggest action at this time or in this state.
		
		The $Mind_i$ server must return *either* an action, or the URL of a server that will generate the action, or a "Cannot suggest action" message.

Get values for this action (How good/bad is this action)	1. mind run ID
	2. x
	3. a
	4. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for this step only, e.g.:
	  1. Q-Temperature
	  2. W-Temperature
		1. Q (How good is this action - How much will this action gain for you)
		2. W (How bad is this action - How much would you pay to stop this and execute your best action instead. How much do you *lose* by having this executed instead of your best action.)
		
		It is possible for an action to have high Q *and* high W.

Inform it about winner (Inform it what happened)	1. mind run ID 2. (OPTIONAL) boolean (whether it was obeyed). Why this might be optional: A Mind server in Hierarchical Q-learning was never even *asked* for an action, so we can't say it was or wasn't obeyed. But we still want to tell it that we took someone else's action ak and got to state y. 3. (OPTIONAL) W (payment to the Mind for losing - see Economy of Mind) 4. (OPTIONAL) Who won (to be precise, the Mind server URL of the winner). For use in Nested systems. 5. (OPTIONAL) ak (the action that was executed). Why this might be optional: If it did not win, it may not *understand* the action that did. But it still wants to know that it did not win. 6. y 7. (OPTIONAL) Score (points scored by action according to World)	1. Confirm 2. (OPTIONAL) Q (how good this was, or how many points scored, for Mind). 3. (OPTIONAL) W (estimate by Mind of how much it lost by this being executed) This command helps the Mind learn how *difficult* it is to win when we are in state x, and how *bad* it is if someone else wins. The Mind may decide to increase the value of W next time round in this state.

Notes:

1. Any Mind server may call another Mind server to get its action. Up until now, the Mind server was not involved in any competition, so it did not have to report to the client that it was calling another server. In response to "Get action", it just calls that other server and returns the action.

   If it *is* involved in a competition, however, it would be far more efficient to postpone calling the other Mind server until it has actually *won* the competition. So in this case it returns the other Mind URL to the client. If it wins, the client can send "Get action" to that other Mind.

9.2.3 Additional Mind_{Feu} queries

A Mind_{Feu} server is a Mind server that accepts *Feudal* commands of the form: "Take me to state c" [PhD, §18.2]. The other Mind servers have their *own* motivations and suggest actions according to them, and clients using them can then decide whether or not to use these suggestions. But a Mind_{Feu} server does not have its own goals, and is only used via this call by another Mind server which has goals:

Request name	Argument data	Return data
Take me to state	1. mind run ID 2. x 3. destination state c 4. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for this step only, e.g.: 1. Q-Temperature 2. W-Temperature	1. (OPTIONAL) a (my suggested action) 2. (OPTIONAL) Mind server URL (of the Mind I want to call now) 3. (OPTIONAL) Q (how good a is for the purposes of getting from x to c) 4. (OPTIONAL) W (how important it is to win the competition now, for the purposes of getting from x to c) The Mind_{Feu} server must return *either* an action or the URL of a server that will generate the action.
How good/bad is this action (to take me to state c)	1. mind run ID 2. x 3. c 4. a 5. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the Mind server. These are parameters for this step only, e.g.: 1. Q-Temperature 2. W-Temperature	1. (OPTIONAL) Q 2. (OPTIONAL) W

9.2.4 Additional Mind_{AS} queries

A Mind_{AS} server is a Mind server that resolves competition between multiple subsidiary Mind servers. Either this is *hidden* from the client (and so the server just appears as an ordinary Mind server above), or else the client provides this list via a special constructor. Having provided the list via the constructor, the client thereafter uses the server just like an ordinary Mind server.

One interesting issue would be, if we have a *hierarchy* of Action Selection competitions, and the Mind_{AS} server will appear as just another primitive Mind_{i} server competing in a *higher-level* Action Selection competition, then where does it get its W-values from? Does it pass upwards the W-values from the winning Mind server *below* it? Of course, interesting as this is, this is a problem for the server author, not an issue for the WWM specification here. The Mind_{AS} server author must somehow *use* the queries defined here to *gather* information from its subsidiary Mind servers to compete at the higher level.

Request name	Argument data	Return data
New run - AS arguments	1. (OPTIONAL) List of Mind server URLs 2. (OPTIONAL) Open-ended set of arguments whose format is interpreted by the AS server. These are parameters for the AS mechanism, e.g.: 1. which of a set of algorithms to use	1. Confirm **or** 1. Refusal (mind URLs not valid, failure in following trail of credit from mind URLs, bad parameters)
Add mind to collection	1. mind run ID 2. Mind server URL	1. Confirm **or** 1. Refusal (mind URL not valid, failure in following trail of credit from mind URL)
Remove mind from collection	1. mind run ID 2. Mind server URL	1. Confirm

A Mind_{M} server may appear with the interface of any of these types of Mind server.

10 How to implement some existing agent architectures as networks of WWM servers

We now show how a number of existing models of agent minds can be implemented as networks of WWM servers using the server queries above.

10.1 Hand-coded program

A hand-coded mind program can clearly be implemented as a single Mind server, receiving `x` and returning `a`. There are a vast number of models of agent mind, whether hand-coded, learnt or evolved, that will repeatedly produce an action given a state. Most of these could be implemented as WWM servers without raising any particular issues apart from having to agree on the format of state and action with the World server.

We will not discuss any of these further, except where they raise particular issues with respect to the WWM. For example, below we will refer in detail to different models of Action Selection, because these raise particular WWM issues.

10.1.1 Initial test - Eliza Mind talks to Eliza World

An initial test of the model could be by connecting two Eliza-type programs together to have a conversation. In this case `x` and `a` are both streams of text. The output `a` for one is the input `x` for the other. Which we regard as the "Mind" and which as the "World" under our scheme does not matter. Even in this initial test we could implement some advanced ideas, such as time-outs, and Mind servers keeping track of previous states. It also raises the issue of how a *human* could become part of the response of a World server or a Mind server.

10.2 The Subsumption Architecture

A Subsumption Architecture model [Brooks, 1986, Brooks, 1991] could be implemented as a hierarchy of Mind_M servers, each one building on the ones below it. Each one sends the current state `x` to the server below it, and then either uses their output or overrides it. So each Mind server sees state `x` and gets to respond. As in Brooks' model, a set of lower layers will still work if the higher layers are removed. On the WWM, there may be many choices for (remote, 3rd party) higher layers to add to a given collection of lower layers.

10.3 Serial models

In serial models, a mind server will "complete" its activity before another mind server will start [Singh, 1992, Tham and Prager, 1994, Wixson, 1991]. This can be driven by a master Mind_M server that passes control from server to server. This Mind_M server needs to know when each goal terminates, which requires it to have a lot of intelligence.

To reduce the demands on the intelligence of the master server, each server *itself* may know whether it is ready to execute or not (preconditions not true yet, or it has just completed its goal). The server can return this in response to the "Ready to suggest action?" query. Then the Mind_M server only needs to know the order of the chain of servers. The servers themselves tell it when it is time to switch to the next server.

Or we could avoid having a master Mind_M server altogether if each server, when its goal is completed, will pass all requests for actions thereafter on to its *successor* server (which it knows about). Then we simply interact with the Society through the *first* mind server in the chain.

10.3.1 Maes' Spreading Activation Networks

Maes' *Spreading Activation Networks* [Maes, 1989, Maes, 1989a] or *Behavior Networks* consist of a network of "servers" which are aware of their preconditions. Servers can be linked to from other servers that can help to make those preconditions come true, or be inhibited by other servers who will cause their preconditions to *not* hold. They can in turn link to other servers whose preconditions their behavior can affect. This might be implemented on the WWM by one server *constructing* the state `x` for the server it is calling, putting the preconditions into `x`.

Maes' *spreading activation mechanism* spreads excitation and inhibition from server to server. This might be implemented on the WWM using the "Change mind strength" message.

10.4 Reinforcement Learning

An ordinary Reinforcement Learning (RL) agent, which receives rewards and punishments as it acts [Kaelbling et al., 1996], can clearly be implemented as a single Mind server. For example a Q-learning agent [Watkins, 1989] builds up Q-values ("Quality"-values) of how good each action is in each state: `Q(x,a)`. This is stored in a data structure inside the agent - either a straightforward lookup table, or else a *generalisation* such as a neural network. Then, given a state, the agent can produce an action based on these Q-values. This maps easily to the WWM model of a Mind server above, as does any similar notion of a state-space learner, e.g. [Clocksin and Moore, 1989].

When learning, the Q-learner can calculate its own reward based on `x`, `a` and `y` [PhD, §2.1.3]. So long as the client informs it what state `y` resulted from the previous action `a`, it can calculate rewards, and learn.

10.5 Hierarchical Q-Learning

Hierarchical Q-Learning [Lin, 1993] is a way of driving multiple Q-learners with a master Q-learner. It can be implemented on the WWM as follows. The client talks to a single Mind_{AS} server, sending it `x` and receiving `a`. The Mind_{AS} server talks to a number of Mind servers. These do not *necessarily* have to support all of the advanced queries of the Mind_{i} server above. They may simply return an action, unaware that there are other minds in the body. The Mind_{AS} server maintains a table of values `Q(x,i)` where `i` is which Mind server to pick in state `x`. Initially its choices are random, but by its own reward function, noting what states the choices take us to, the Mind_{AS} server fills in values for `Q(x,i)`. Having chosen `i`, it passes on the action suggested by Mind server `i` to the client.

In fact, to save on the number of server queries (which is a more serious issue on the WWM than in a self-contained system), we would do the following. Each time step, when presented with a state `x`, the Mind_{AS} server makes a decision based on its Q-values (initially random) and then, having picked action `i`, queries a *single* Mind server `i` for its action. Note then that the question of being obeyed or not obeyed does not apply to the other Mind servers - they were not even *asked* for an action on this step.

Whether they were asked for an action or not, the other Mind servers can still *learn* while in this system, if the Mind_{AS} server tells them what action it executed. i.e. They will need to support at least one of the advanced Mind_{i} queries above. In general, any Mind server in a competition needs to be informed if it was obeyed, and what action was taken. Otherwise it may think that its action (which was not taken) led to the new state.

One interesting possibility with Hierarchical Q-Learning on the WWM is that the Mind_{AS} server *need not know its list of Mind servers in advance*. It can be passed this list by the client at startup, using the special constructor defined above. Of course then it will have to learn from scratch (i.e. it is sent a high Temperature parameter).

Another possibility is that the subsidiary Mind servers *need not be Q-learners*. They could be any type of Mind server, and the Mind_{AS} server simply learns which one to let through.

10.6 Action Selection with a single query or multiple queries

For many of the following models it will be useful to distinguish between two types of Mind_{AS} server:

1. An $\textbf{AS}_{\textbf{s}}$ **server** makes a single query of each Mind server before making its decision.

2. An $\textbf{AS}_{\textbf{m}}$ **server** makes multiple queries of each Mind server before making its decision.

Hierarchical Q-Learning is not either of these because it does not even query *all* Mind servers *once*. Based on its `Q(x,i)` values it just makes one query of one Mind server.

10.7 Static measures of W

We will consider a number of schemes where Mind servers promote their actions with a weight W, or "W-value". Ideally the W-value will depend on the state `x` and will be higher or lower depending on how much the Mind server "cares" about winning the competition for this state [PhD, §5].

A *static* measure of the W-value [PhD, §5.3] is one in which the Mind server promotes its action with a value of W based on internal reasons, and independent of the competition. Any such method (including, say, W=Q) can clearly be implemented as a Mind_{i} server. There will be a number of Mind_{i} servers, and then a simple Mind_{AS} server which lets through the one with the highest W-value. This is an AS_{s} server.

10.8 Dynamic measures of W

A *dynamic* measure of W [PhD, §5.5] is one in which the value of W changes depending on whether the Mind server was obeyed, and perhaps on who won instead if it did not. Clearly this is an AS_s server that queries once, lets through the highest W, and then *reports back* afterwards to each server whether or not it was obeyed, using the WWM commands defined above. The server may then *modify* its W-value next time round in this state.

10.9 W-learning

W-learning [PhD, §5, §6] is a form of dynamic W where W is modified based on (i) whether we were obeyed or not, and (ii) what the new state y is as a result. This can clearly be implemented on the WWM as an AS_s server. All the variations, such as *Stochastic* highest W [PhD, §6.5], and the winner *not* altering their W-value [PhD, §6.3], can clearly also be implemented using the WWM queries defined above.

In the pure form of W-learning [PhD, §6, §11, §13] the Minds do not even share the same suite of actions, and so, for example, cannot simply get together and *negotiate* to find the optimum action (see below). The inspiration was simply to see if competition could be resolved between Minds that had as little in common as possible. I was unable to give convincing examples where this might arise. Now with the WWM, and Minds coming from totally different origins, I hope it is clearer what the usefulness of this is. This is the type of AS method we will need for many situations on the WWM.

10.10 Strong and Weak Mind servers

[PhD, §8] showed how altering the absolute size of a Q-learning Mind server's rewards can change the size of the W-values it presents, *without* altering its policy. To be precise, we can multiply all the base Q-values by any constant `c` to produce an agent with the same policy but different W-values. As a result one could ask for a "strong" version of a Mind server, which would have the same policy as a weak version, but present larger W-values. This would be done by presenting the "mind strength" constant `c` as an argument to the server at startup. For a full explanation of how to carry out "Normalisation" and "Exaggeration" of the same basic behaviour, see [PhD, §C, §D]. Artificial Selection could search for good combinations of strong and weak servers by either:

1. Automated search.

   or:

2. By hand. Slowly increase or decrease the strength values, leaving all the *details* of the competition to be resolved automatically, and then observe the resulting global behaviour [PhD, §16.3, §17.2].

10.11 Matching World state definition with Mind state definition

Reinforcement Learning also shows us how we can get away with not defining a format for the state `x` and action `a`. In RL, `x` and `a` are abstractions, so that, for example, we define a model where executing action `a` in state `x` leads to state `y` with probability $P_{xa}(y)$ - yet there is no need to actually define the format of `x` and `a` at this point.

Similarly with the WWM we leave state and action as undefined streams of text data terminated by `</data>`. How these streams are to be decoded is a matter for the World and Mind servers to agree among themselves. Servers will advertise (at their URLs) what format they expect and what format they generate, and others will act accordingly. Collections of servers that have incompatible formats, and therefore do not work, are not a problem. People will *expect* that vast numbers of Societies will not work at all, or work poorly, and the whole mindset will be to *search* for ones that work better than others, follow "Top 10" lists of good performers in a certain World, and so on. Societies that are not compatible, or even ones that *are* compatible but work *poorly*, will simply not be advertised.

10.11.1 "Islands" of compatible worlds

This does raise the question, though, of whether different *sub-zones* of the WWM will develop, each incompatible with the other. It seems that this will indeed happen. For any World, there will be an island of Minds that understand this World and interpret its definition of state `x` or some subset of `x`. If the World is popular, other Worlds might be built to the same specification, so that the same Mind can act in all of these Worlds [Ray, 1995]. There will be a (perhaps very large) "island" of compatible Worlds and Minds, separate from other islands built to different specifications.

The AS servers might be more independent of the World definition, so that the same AS server can be used in different "islands". The AS server will receive `x` and return `a`, but need not understand the structure of either, but just pass on `x` to the Mind servers that *do* understand it, make a decision as to who to pick based on, say, the highest W-value, and then return whatever meaningless stream of data they provide as the action `a`.

10.11.2 The "island" of the physical world

For real robots, since the real physical world is the same for everyone, one might think there would be just one island - so that any real-world Mind could act on any real-robot World server. Not so, of course, because how you sense the real world (state `x`) depends on what sensors the robot hardware possesses, and what format they deliver their input in. One could imagine, though, that there will be a separate island clustered around each robot *make*. For instance, Mind servers that will run on any *Khepera* robot. Mind servers that will run on any LEGO Mindstorms robot. Mind servers that will run on a Nomad robot that has certain specified add-ons.

So, in conclusion, the network of World-Wide-Minds will not be unified, but will consist of a number of separate incompatible islands.

10.11.3 Mind servers with different senses in the same Society

[PhD, §6.6] discussed where Mind servers may have different senses, even within the *same Society*, which makes their competition even more confusing. Sometimes in a particular state they win, and sometimes, in what seems to be exactly the same state (but is perceived by another Mind server as a different state) they lose (because the other Mind server competed differently).

In a WWM implementation of this, the Mind_{AS} server may receive the *full* state from the World, and then send a different sub-space of that to each Mind server as *its* input state. This is actually what we did with Hierarchical Q-learning [PhD, §4.4]. Both W-learning with subspaces [PhD, §7, §8] and W-learning with full space [PhD, §10] can also clearly be implemented as AS_{s} servers using the WWM primitives above.

10.12 Global Action Selection decisions

If Minds *do* share the same suite of actions, then we can make various global decisions. Say we have `n` Mind servers. Mind server `i`'s preferred action is action a_i. Mind server `i` can quantify "how good" action `a` is in state `x` by returning:

$$Q_i(x,a)$$

and can quantify "how bad" action `a` is in state `x` by returning:

$$Q_i(x,a_i) - Q_i(x,a)$$

Then we have 4 basic approaches [PhD, §14]:

1. **Maximize the Best Happiness**:

$$MAX_a \; MAX_i \;\; Q_i(x,a)$$

which is in fact the same as static W=Q above, and can be implemented as an AS_s server, with just one query to each Mind server to get its best action and its Q-value.

2. **Minimize the Worst Unhappiness**:

$$MIN_a \; MAX_i \;\; (Q_i(x,a_i) - Q_i(x,a))$$

which is an AS_m server, requiring multiple queries of each Mind server.

3. **Minimize Collective Unhappiness**:

$$MIN_a \;\; [SUM_i \; (Q_i(x,a_i) - Q_i(x,a))]$$

which is an AS_m server.

4. **Maximize Collective Happiness**:

$$MAX_a \;\; [SUM_i \; Q_i(x,a)]$$

which is an AS_m server.

10.13 Other Action Selection methods based on RL

There are a number of other related AS methods, which can all be implemented as WWM servers:

1. **Negotiated W-learning** [PhD, §11] is an AS_m method.

2. **Collective W-learning** [PhD, §12.2] can be implemented by the $Mind_{AS}$ server building up a table of `W(x,i)` - the combined *loss* that Mind server `i` causes for *everyone else* when it wins (each server reports back to the $Mind_{AS}$ server their own loss W). Like Hierarchical Q-learning, this is neither AS_s nor AS_m. It chooses a Mind server based on its W-values table and then makes one query of one server to return its action. Then it sends a command to *every* server to tell it what happened, and adjusts its W-value according to the losses the servers report in their responses. All the variants of this can clearly be implemented as well, including Stochastic lowest W [PhD, §12.2.2] and **Negotiated Collective W-learning** [PhD, §12.2.4] (which is an AS_m method).

3. **Collective Equality** [PhD, §12.4] is an AS_m method.

4. Any form of **scaling the W-value** [PhD, §8.1.2] can be implemented as well, with extra complexity in the Mind server, but no need for extra server queries.

5. As referenced in [PhD, §F], there are other measures of Happiness and Unhappiness that may or may not make sense, all of which can be implemented by single or repeated WWM queries.

10.14 Other parallel models

The DAMN Architecture [Rosenblatt, 1995, Rosenblatt and Thorpe, 1995] implements an action selection method similar to Maximize Collective Happiness. The Q-values of Mind server `i` are multiplied by weights w_i which reflect the current priorities of the system. This could be implemented as an AS_m server, where the AS server maintains a set of weights w_i.

Product Maximize Collective Happiness [PhD, §15.5.2], adapted from Grefenstette's work [Grefenstette, 1992], can be implemented on the WWM as an AS_m server.

A number of other authors [Aylett, 1995, Tyrrell, 1993, Whitehead et al., 1993, Karlsson, 1997, Ono et al., 1996] implement, using a variety of notations, one of the 4 basic AS methods defined above [see PhD, §15.4]. Though none, as far as I am aware, have tried a Minimize the Worst Unhappiness strategy.

10.15 The AS server remembering the winner

To reduce the number of server queries needed, the AS server may *remember* who won the state last time, and build up a table `k(x)` for the winner (or `a(x)` for compromise actions) so it does not have to run the competition again [PhD, §11]. Obviously this only works if the servers remain unchanged (they are not learning), and if the *collection* of servers remains unchanged (no new servers, or old ones leaving).

10.16 Dynamically changing collections

To continue that last point, the WWM server queries allow for a collection of Mind servers where new ones are added or old ones removed during the course of the run [PhD, §17.6]. The implementation of the "Add mind" and "Remove mind" commands in the Mind_{AS} server will then send a "Collection changed" message to all of its subsidiary Mind servers, to inform them that the competition has changed and they may have to re-learn their W-values.

10.17 Nested Mind servers

Digney [Digney, 1996] defines Nested Q-learning, where *each* Mind in a collection is able to call on any of the others. Each Mind server has its own set of actions $Q_i(x,a)$ *and* a set of actions $Q_i(x,k)$ where action `k` means "do whatever server `k` wants to do" (as in Hierarchical Q-learning). Of course we already have in general that a Mind_{M} server can call other Mind servers. What is different here is:

1. It *learns* how good it is to call other servers. To do this, it needs to be supplied with some extra information, such as who won.
2. Because it learns, it can be supplied with the list of Mind servers at startup, rather than having it pre-coded.

In a WWM implementation, each Nested server has a list of Mind URLs, either hard-coded or passed to it at startup. So the Nested server *looks* like a Mind_{AS} server co-ordinating many Mind servers to make its decision. But of course it is *not* making the final decision. It is merely *suggesting* an action to the master Mind_{AS} server that coordinates the competition between the Nested servers themselves. When the master Mind_{AS} server is started up with a list of Mind servers, it passes the list to each of the servers.

Consider the number of server queries in a Nested WWM system. The master Mind_{AS} server is given `x` and asked for an action `a`. It sends `x` to each Mind server. Server `i` looks at its Q-values and either suggests an action directly, or returns the URL of some server `j`. Server `j` is *not* yet queried. We wait to see if server `i` can win the competition. (In fact, server `j` may already have been queried separately for its *own* action.) If server `i` wins, then we query server `j` and get an action, which may be "Do what server `k` does" and so on. As well as allowing the Nested server to return a Mind URL instead of an action, we also need the master server to *tell it* the Mind URL of the winner. Remember that in Hierarchical Q-learning, the master server needs to know who won, so it can put values on Q(x,i). But of course it knows itself who won. Here, the Nested server needs to know who won, so it can put values on $Q_i(x,k)$. So it needs to be *told* who won by the master server.

10.17.1 Each server calling a different list of servers

In the basic model we just described, all Mind servers can call all other Mind servers in the list. But in fact, the list could be different for each server. Each server could hard-code its own list of servers that it may call, similar to how any hand-written Mind_{M} server hard-codes its list of servers. One confusion would be, when we tell the server who won, and we pass it the URL of a server that is not in its list of possible servers to call.

10.17.2 Servers outside the AS loop

[PhD, §18.1] also shows how some of the Nested servers might actually be outside the Action Selection competition, and simply *wait* to be called by a server that *is* in the competition. I call these "passive" servers. We have the same with hand-coded Mind_{M} servers, where some Mind servers may have to wait to be called by others. A server may be "passive" in one Society and at the same time "active" (i.e. the server is in the Action Selection loop) in a different Society.

10.18 Feudal Mind servers

Watkins [Watkins, 1989] defines a *Feudal* (or "slave") Q-learner as one that accepts commands of the form "Take me to state `c`". On the WWM, these Feudal Mind servers will be driven by other Mind servers that actually have preferences about what goal state to get to. In Watkins' system, the command is part of the current state. Using the notation `(x,c),a -> (y,c)` the slave will receive rewards for transitions of the form: `(*,c),a -> (c,c)` So the master server drives the slave server by *explicitly* altering the state for it. We do not have to change our definition of the server above. It receives `x` and produces `a`. It is just that the server driving it is constructing the state `x` rather than simply passing it on from above.

Another possibility is that the real state and the command are *explicitly* separated in the server query, which is what we allowed for with the additional Mind_{Feu} queries above. Using either of these approaches, the WWM model allows for Mind servers that provide a service of taking one to explicit goal states. e.g. Moore [Moore, 1990] has a concept of an explicit goal state, and Kaelbling's Hierarchical Distance to Goal (HDG) algorithm [Kaelbling, 1993] addresses the issue of giving the server *new* explicit goal states at run-time.

10.19 The sub-symbolic Society of Mind

The Nested and Feudal models are combined in [PhD, Fig. 18.4] showing the general form of a Society of Mind based on Reinforcement Learning. It is suggested that Reinforcement Learning would be one of the most fruitful areas in which to begin implementing the WWM. That is, we shall begin with a *sub*-symbolic Society of Mind. Indeed, the whole model of a complex, overlapping, competing, duplicated Society of Mind that we have developed in this paper is based on the generalised form of a Society of Mind based on Reinforcement Learning.

10.20 More complex communication between Mind servers

Baum's Economy of Mind [Baum, 1996] has new Mind servers paying off the old ones to gain control. This can still be done through our model. The payments would be managed through the Mind_{AS} server, which receives payments through the W-value, and redistributes them through the "Inform it about winner" command.

10.21 Is this a sub-symbolic model?

So far we have only defined a protocol for conflict resolution where the AS server makes queries of the Minds for different numeric weights, e.g. "How much will you pay to stop this happening?". As discussed, we may need further protocols for more sophisticated, symbolic communication among Mind servers. We imagine that numeric weights will be easily generated by *sub-symbolic* Minds, and are harder to generate in symbolic Minds. This is because symbolic Minds often know *what* they want to do but not "how much" they want to do it. Sub-symbolic Minds, who prefer certain actions precisely because numbers for that action have somehow risen higher than numbers for other actions, may be able to say precisely "how much" they want to do something, and *quantify* how bad alternative actions would be [PhD, §5.2].

The Action Selection model may be sometimes limiting, though, in *demanding* that the Mind win the competition objectively. We cannot just *say* that "Mind M3 should always win when the World is in state Z". Instead, M3 has to *win the competition* in that state. The solution then is we can surround the Action Selection by a Mind_{M} server that *ensures* that M3 wins in state Z.

It may be that in the *symbolic* domain we will make a lot more use of Mind_M servers, and maybe even avoid Action Selection altogether. This might be a popular alternative to having Minds generate Weights to resolve competition. The drawback, of course, is that the Mind_M server needs a lot of intelligence. It needs to understand the goals of all the Mind servers. This relates to the "homunculus" problem, or the need for an intelligent headquarters, see [PhD, §5].

11 HTTP CGI using XML

We now ask what actual technology should we use to implement the WWM queries. I suggest one overriding objective:

1. *That the WWM server authors be required to know as little as possible to get their servers on the network.*

The server authors are interested in AI, not necessarily in networks. They may only know AI programming languages such as LISP. They may have never written a network application, and they may not want to learn. If we accept this criterion, then we should seek a *lowest-common-denominator* approach that will enable AI researchers to put their minds and worlds online with the minimum of delay. Ideally, we would have the following:

1. The WWM server authors can write their program in any programming language, according to any programming methodology, on any operating system.
2. The WWM server authors do not have to install any new software, but can run their programs on existing Web servers.
3. World server authors do not have to learn any particular language for describing a world, such as VRML.
4. The WWM server authors do not have to learn any new programming language, such as Java, Perl, or any other language.
5. The WWM server authors do not have to learn any new network or object-oriented programming techniques. If all they know how to do is read from stdin and write to stdout, that should be sufficient to write a WWM server.

As [Bosak and Bray, 1999] put it: *"schemes that rely on complex, direct program-to-program interaction have not worked well in practice, because they depend on a uniformity of processing that does not exist."* [Paulos and Canny, 1996] make a strong call for a lowest-common-denominator approach in remote access to robots.

11.1 HTTP CGI

It is clear what the lowest-common-denominator system is on the network today, the system by which thousands of programmers have put programs and scripts online that were never online before. It is CGI. It is proposed that the lowest-common-denominator implementation of the WWM be done using CGI across HTTP. Every AI programmer has access to a HTTP server with CGI, and every AI programmer can write a program that receives stdin and writes to stdout.

11.2 XML

The data transmitted would not be HTML, as in "normal" CGI scripts, but would rather be the server queries, responses, and associated data. It is proposed that this be encoded as text-based XML [Bosak and Bray, 1999] rather than in a binary format. The advantages would be:

1. XML is human-readable text, so it can be read and altered by hand on any system. One does not have to go through any particular application. In particular, it is easy to get your own program (in any language) to generate XML text output.
2. As for reading XML input, it is a standard format (open tag, close tag) so that XML parsers are available in most languages (and it is easy to parse yourself in any case).
3. It can be transmitted by CGI now to and from existing Web servers, with no extra modification needed.

4. It is easy to extend the tag definitions (and hence the server query definitions) in future, *without* breaking the old definitions.

11.2.1 XML encoding of server queries

An example of an XML-encoded query would be as follows. The client asks the mind server to set up a new run, telling it what world we will be running it in. HTTP CGI POST is used to send the data to the mind server, so that the mind server receives the following XML code on stdin:

```
<xml>
<query name="New run">
<data name="world run ID"> 40031 </data>
<data name="world display URL"> http://worldserver/currentruns/40031.html </data>
</query>
</xml>
```

The mind server replies, assigning the client a unique run ID. The server writes to stdout:

```
<xml>
<response name="New run">
<data name="mind run ID"> 5505 </data>
<data name="mind display URL"> http://mindserver/currentruns/5505.html </data>
</response>
</xml>
```

The client uses this unique ID in each subsequent request:

```
<xml>
<query name="Get action">
<data name="mind run ID"> 5505 </data>
<data name="state">
x
</data>
</query>
</xml>
```

The state is simply a series of text characters representing the state, terminated by the `</data>` tag. How to decode these characters is something that the servers have to agree among themselves. The mind server returns an action:

```
<xml>
<response name="Get action">
<data name="action">
a
</data>
</response>
</xml>
```

Again, the format of the action is something the servers have to agree on.

11.2.2 "AIML"

When we have precisely expressed the server queries as XML, we might call the resulting markup language **"AIML"** or "AI Markup Language".

Though it has to be noted that the name "AIML" is currently being used in a much more restricted domain - It is being used by the ALICE chatbot project [alicebot.org] as a means of allowing users to define their own chatbots. In this case, XML is being used to define the *program* itself rather than the data being passed back and forth. A chatbot is only one form of many possible types of WWM server, so a name like "ChatML" or even "ElizaML" would have been much more appropriate than "AIML". But criticism of this point aside, one interesting thing about ALICE is that in trying to allow non-technical users construct entire servers, in some ways it takes 3rd party involvement even further than envisaged in this work.

The *Swarm* project, which aims to provide a standardised simulator for multi-agent worlds, has also recently moved to XML because of the restrictions inherent in forcing the use of any particular programming language [Daniels, 1999]. Again, XML is used as a way of defining the environment and agents and *avoiding programming*, rather than as a way of communicating between separate minds and worlds. [Noda et al., 1998] is probably the closest previous work to the WWM, using a client-server design, programming language independent, transmitting ASCII strings.

11.3 Addressing

All requests to a WWM server are requests to a CGI program on a Web server:

```
http://site/directory/program
```

All arguments (including the type of WWM request being sent) are passed in stdin.

11.4 Persistent CGI

Normally, CGI is 1 process per request. In a normal CGI request, input comes in on stdin, *a new process is started*, output is generated on stdout, and then *the process terminates*. Here, though, we have a WWM server program where we want to send it *multiple* requests at different times over the course of a long run. The question is: How do we maintain state in between requests?

1. **Save to disk and Restore -** The first issue is whether we can maintain state *at all*. The lowest-common-denominator approach would be for the programmer to *save the state* of their program to disk after each WWM request, and then *restore* it (from disk) when the next request comes in. All programmers can do this, though it may be some work. It is also very inefficient, starting the program again from scratch for each new request. But with powerful machines, this might not matter greatly. The important point is that the AI programmer can save and restore the state without learning any new programming techniques.

2. **"Persistent CGI" -** The second issue is whether an *efficient* WWM server can be built. For example, one where the "Start run" request starts a process running that does *not* terminate when the "Start run" request terminates. Later CGI requests simply talk to this independently-running process. Finally, the "End run" CGI request terminates the process.

   It is no surprise that **this is actually a well-known issue with CGI programs,** and there are many approaches to it discussed in the CGI community, especially where scripts talk to large *databases*. How exactly to do it depends on the programming language and server used, and **there is no standard solution.**

It is not the job of the WWM to propose a solution to the problem of persistent CGI. WWM server authors should be able to use any of the technologies that the CGI community propose. Here we simply note that for an *efficient* WWM server the server author may need to learn some network programming, which is something we wanted to avoid. But this is not necessary to set up a WWM server *at all*, which can be done using the save-to-disk method.

The AI programmer might get the server online *initially* using save-to-disk, and maybe sometime later, after learning some network programming, convert to a persistent process. The client would not notice any difference (except a speed improvement).

11.5 Asynchronous worlds

The CGI model is client-driven. Servers only respond to client requests. So how could we have an *asynchronous world* - a World that changes even when no client is making requests to it? Note this is actually the same issue as persistent CGI above: How can we start a process that does not end when the "Start run" request ends, but that carries on, and only ends when a new "End run" request is made?

It is also the same issue as: If the client *never* issues an "End run" command, can we *time it out* if we haven't received queries from it in a long time. This would be important to relinquish control of a shared resource, e.g. a robot [Stein, 1998]. This could still be client-driven: The old client gets timed out when the new client makes its "Start run" attempt.

There is a similar "quick and dirty" way of making an asynchronous world while still retaining the simple client-driven model: The *next* time the client makes a request, the server calculates the time since the last request, and runs the world forward that number of timesteps *before* replying. An alternative, but more complex way, would be for the server to have its own "clock-tick" client, which sends it a periodic clock tick using the "no op" command. Each time it receives a clock tick it can update the world.

Part 4 - Future work and Conclusion

12 Future work

This is clearly the start of an enormous program of implementation and testing. The major immediate issues to be solved are the following:

12.1 Define the server queries

1. Nail down the list of server queries and responses above. Is this list sufficient to implement all current sub-symbolic agent minds and worlds?
2. **"AIML" -** Precisely express these server queries and responses in XML. Define which fields are mandatory and which are optional. Define all error conditions, error codes, error text messages, etc.

12.2 Define the client user view

Having defined the server queries and encoded them in XML, **server authors** can now construct servers, write programs to talk to servers, display the results, etc. So in that sense, our work is done.

But how about the **client user**? How could they use the system, without having to write programs to talk to servers? Obviously some client software is needed. The basic question is whether this software can be provided through existing Web browsers or whether some plug-in or separate client application needs to be installed by each user.

12.2.1 Client use through existing Web browsers

Consider that we want the non-technical client user to be able to do the following:

1. Browse existing Mind and World servers.
2. Specify a Mind server with a number of arguments (some of which may be the addresses of other Mind servers), and a World server with a number of arguments.
3. Be able to *link* to this Mind/World combination (with arguments) from a Web page, so that any 3rd party may run it.
4. Run this Mind/World combination.
5. View the run in progress. 3rd parties can also view the run in progress.
6. The run either lasts for a fixed time, or until the user clicks some "End run" button.

This could in fact all be done in existing Web browsers, if we can assume the existence of an Internet portal to help the non-technical user:

1. Servers have URLs at which they describe what they do.
2. A public site, that we shall call `WWM.COM`, helps clients construct WWM combinations. It provides a HTML form in which one can specify the two remote servers, plus arguments for them.
3. When you complete the form, it sets up a *client URL* at `WWM.COM` explaining the arguments and what servers are being used, and providing a "Start run" button.

4. When you (or any 3rd party) click the "Start run" button on this page, a special *client program* starts. This program will implement some algorithm making requests to the remote Mind and World servers, managing time-outs, repeated queries, and so on. Perhaps this program actually runs on `WWM.COM`'s machine. Or perhaps it is a Java applet written by `WWM.COM` and running on the client machine.

5. Once the client user clicks the "Start run" button, `WWM.COM` directs us onto a new page, specific to this particular run:

   ```
   http://WWM.COM/user-no-5503/combination-no-7/run-no-20
   ```

   This provides links to the World display URL, Mind display URLs, etc. This run can be viewed by any 3rd party.

6. Either the client program runs for a fixed number of steps and then terminates (in which case it could appear to the client user as a normal CGI script, albeit a very long one), or else clicking the button starts a program that does not terminate. In the latter case `WWM.COM` provides an "End run" button perhaps in the Java applet or perhaps on the page above. When clicked, this terminates the program that "Start run" began. In fact we want other people to be able to view the run at the URL above, but only the client user that started the run (which is *not* necessarily user-no-5503) can terminate it. In which case we store information on the client side, e.g. as a cookie, or perhaps have a special URL:

   ```
   http://WWM.COM/user-no-5503/combination-no-7/run-no-20/uniqueid
   ```

   Presumably the client user can bookmark this, log out, and wait for *weeks* before coming back and hitting "End run".

`WWM.COM` will have pages listing of all the combinations constructed on it, perhaps sorted by World. Using the "End run" information, `WWM.COM` may be able to *highlight* combinations that scored well, run "Top 10" lists and so on.

Servers have URLs as well, and each server could have links to all the servers it calls, all the servers that call it, all the combinations it was ever part of (i.e. it would point to many different client URLs around the network), and which of these combinations scored well.

12.2.2 Dedicated client software

While it should be possible to use the WWM through a normal Web browser, it should be possible to write a dedicated client application that makes using it *easier* for non-technical client users (and probably server authors too). Ideally this software would still allow us, when we are finished, to *link* to a successful Combination that can still be run by someone who does not have the software. Also conversely, clicking on a Combination in our Web browser would launch this software.

12.3 Testing

To fully define the above, we should build and test many different types of WWM networks:

1. Implement all of the agent minds and worlds mentioned in this paper, including Tyrrell's world, Noda's soccer server, a generic Q-learning mind, generic Action Selection servers, etc.

2. Implement all of the various alternative forms of WWM servers and networks described in this paper - Mind_{M} servers, Mind_{AS} servers, World_{W} servers, multi-level networks, real robots, shared virtual worlds, world display URLs (for virtual worlds and real robots), time-outs, synchronous and asynchronous worlds, learning servers, automated searches with no user interface, evolutionary searches, etc.

3. Take existing AI projects that have already been put online [see Yahoo list of AI programs online] and put them online as WWM servers - including Eliza, MGonz, etc.

12.4 Long-term prospects

If all of the above is solved, we should have a scheme for implementing the full promise of the WWM for all sub-symbolic models of mind, and many simple symbolic ones. What, then, is the next step?

1. Implement more complex *symbolic* models of mind as networks of WWM servers. Presumably these will be more *high-bandwidth* than the schemes discussed in this paper. The servers will communicate by some *agent language*, rather than just by numeric values. As discussed above, standardising a high-level agent language is difficult.

   Within symbolic AI, DPS uses multiple *co-operating* minds to solve one problem. We can currently implement this on the WWM using a top-level Mind_{M} server to collate multiple Mind servers' results to solve a problem. There is scope, though, for expanding the list of server queries to more efficiently implement some of the DPS models, such as blackboard architectures and contract nets.

2. Apply the model to more than just *behaviour*. Can we share / use / partially-override other people's *representations*? There may be applications to *distributed memory across the network* [Porter et al.], or *symbol-grounding* using multiple representations. Note that we can already implement something like this with the current system, where the same input goes to multiple Mind servers, each of which may form its own representations. Taking this further might involve a representation spread over multiple servers, or servers passing representations to each other.

3. Servers might send signals to each other that are slightly independent of the current state. For example, a flow of *emotions* or *hormones* or excitation or inhibition flowing through the network [e.g. Maes, 1989, Maes, 1989a]. Perhaps some of this could be done with the "Change mind strength" message above.

   Or one could have servers constructing new plans and thoughts independent of the current state. Perhaps "consciousness" servers constructing a flow of *narratives* through the network.

4. Can we combine high-bandwidth (symbolic) and low-bandwidth (numeric) servers in networks?

5. Minds talking to Minds. - As discussed, we may have a mind server talking to another mind server as its "world". In this case we could have "real" societies of individuals instead of just societies of mind. This may relate to experiments in autonomous language formation [Steels and Kaplan, 1999], transmission of memes [Dawkins, 1976], and so on. In fact, the first test of this system will probably be in the domain of language evolution [Walshe, 2001].

13 Conclusion

There are *two* issues here - first, that we need a system of decentralised network AI minds, and second a proposed protocol for it. Even if the protocol here is not adopted, the first part of this paper (the need to decentralise AI) stands on its own.

13.1 Endnote - Showing the world what a mind looks like

If the WWM scheme becomes successful, much of the user population of the Internet will gradually become familiar with minds made up of hundreds or even thousands of distributed components; minds that have little identifiable headquarters, but are made up of a crowded collection of sub-minds, duplicating, competing, overlapping, communicating and learning, with *"alternative strategies constantly bubbling up, seeking attention, wanting to be given control of the body"* [PhD, §18.3].

Such models may be long familiar (at least in theory) to AI researchers, but they are not much understood outside of AI. As a result, outside of AI, people still tend to judge statements such as "The mind is a machine" by the standards of machines *they* are familiar with, such as, say, Microsoft Word. The WWM scheme may help large numbers of people expand their imagination to think about what a mind *could* be.

14 Acknowledgements

Thanks to my brother Richard for encouraging me to pursue this project.

Thanks to Dave O'Connor, Ciaran O'Leary, Amanda Marsh, David Sinclair and Ray Walshe for important contributions and discussions.

Thanks to Elizabeth, Thomas and James for putting up with my absences.

References

Aylett, Ruth (1995), Multi-Agent Planning: Modelling Execution Agents, *Papers of the 14th Workshop of the UK Planning and Scheduling Special Interest Group*.

Axelrod, Robert and Hamilton, William (1981), The evolution of cooperation, *Science* 211(4489):1390-6.

Axelrod, Robert (1984), *The evolution of cooperation*.

Baum, Eric B. (1996), Toward a Model of Mind as a Laissez-Faire Economy of Idiots, *Proceedings of the Thirteenth International Conference on Machine Learning*.

Blumberg, Bruce (1994), Action-Selection in Hamsterdam: Lessons from Ethology, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB-94)*.

Bosak, Jon and Bray, Tim (1999), XML and the Second-Generation Web, *Scientific American*, May 1999.

Brooks, Rodney A. (1986), A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* 2:14-23.

Brooks, Rodney A. (1991), Intelligence without Representation, *Artificial Intelligence* 47:139-160.

Brooks, Rodney A. (1997), From Earwigs to Humans, *Robotics and Autonomous Systems*, Vol. 20, Nos. 2-4, June 1997, pp. 291-304.

Brooks, Rodney A.; Breazeal, Cynthia; Marjanovic, Matthew; Scassellati, Brian and Williamson, Matthew M. (1998), The Cog Project: Building a Humanoid Robot, *Computation for Metaphors, Analogy and Agents*, Vol. 1562 of Springer Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998.

Brooks, Michael (2000), Global Brain, *New Scientist*, 24th June 2000.

Bryson, Joanna (2000), Cross-Paradigm Analysis of Autonomous Agent Architecture, *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)* 12(2):165-89.

Bryson, Joanna (2000a), Hierarchy and Sequence vs. Full Parallelism in Reactive Action Selection Architectures, *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior (SAB-00)*.

Bryson, Joanna; Lowe, Will and Stein, Lynn Andrea (2000), Hypothesis Testing for Complex Agents, *Proceedings of the NIST Workshop on Performance Metrics for Intelligent Systems*.

Clocksin, William F. and Moore, Andrew W. (1989), Experiments in Adaptive State-Space Robotics, *Proceedings of the 7th Conference of the Society for Artificial Intelligence and Simulation of Behaviour (AISB-89)*.

Daniels, Marcus (1999), Integrating Simulation Technologies With Swarm, *Proceedings of the Workshop on Agent Simulation: Applications, Models, and Tools*, University of Chicago, Oct 1999.

Darwin, Charles (1859), *The Origin of Species*.

Dawkins, Richard (1976), *The Selfish Gene*.

Diamond, Jared (1997), *Guns, Germs and Steel: The Fates of Human Societies*, Jonathan Cape, London.

Digney, Bruce L. (1996), Emergent Hierarchical Control Structures: Learning Reactive/Hierarchical Relationships in Reinforcement Environments, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*.

Ginsberg, Matthew L. (1991), Knowledge Interchange Format: The KIF of Death, *AI Magazine*, Vol.5, No.63, 1991.

Goertzel, Ben (1996), *The WorldWideBrain: Using the WorldWideWeb to Implement Globally Distributed Cognition*, goertzel.org/papers/wwb.html

Goldberg, Ken; Santarromana, Joseph; Bekey, George; Gentner, Steven; Morris, Rosemary; Sutter, Carl and Wiegley, Jeff (1996), A Tele-Robotic Garden on the World Wide Web, *SPIE Robotics and Machine Perception Newsletter*, 5(1), March 1996.

Goldberg, Ken; Chen, Billy; Solomon, Rory; Bui, Steve; Farzin, Bobak; Heitler, Jacob; Poon, Derek and Smith, Gordon (2000), Collaborative Teleoperation via the Internet, *IEEE International Conference on Robotics and Automation (ICRA-00)*.

Grefenstette, John J. (1992), The Evolution of Strategies for Multi-agent Environments, *Adaptive Behavior* 1:65-89.

Harvey, Inman; Husbands, Philip and Cliff, Dave (1992), Issues in Evolutionary Robotics, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB-92)*.

Heylighen, Francis (1997), Towards a Global Brain: Integrating Individuals into the World-Wide Electronic Network, published in German in Brandes and Neumann, eds., *Der Sinn der Sinne* (Steidl Verlag, Gottingen), but available in English at: pespmc1.vub.ac.be/papers/GBrain-Bonn.html

Holland, John H. (1986), Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms applied to Parallel Rule-Based Systems, in Michalski et al., eds., *Machine Learning: an Artificial Intelligence approach*, Vol.2.

Humphrys, Mark (1989), The MGonz program, www.compapp.dcu.ie/~humphrys/eliza.html

Humphrys, Mark (1991), *The Objective evidence: A real-life comparison of Procedural and Object-Oriented Programming*, technical report, IBM Ireland Information Services Ltd.

Humphrys, Mark (1997), *Action Selection methods using Reinforcement Learning*, PhD thesis, University of Cambridge, Computer Laboratory (Technical Report no.426), www.compapp.dcu.ie/~humphrys/PhD

Humphrys, Mark (1997a), *AI is possible .. but AI won't happen: The future of Artificial Intelligence*, the "Next Generation" symposium, the "Science and the Human Dimension" series, Jesus College, Cambridge, Aug 1997.

Humphrys, Mark (1999), Why on earth would I link to you?, *The Irish Times*, 15th Feb 1999.

Hutchens, Jason (undated), *How MegaHAL works*, amristar.com.au/~hutch/megahal/How.html

Kaelbling, Leslie Pack (1993), Hierarchical Learning in Stochastic Domains, *Proceedings of the Tenth International Conference on Machine Learning*.

Kaelbling, Leslie Pack; Littman, Michael L. and Moore, Andrew W. (1996), Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research* 4:237-285.

Karlsson, Jonas (1997), *Learning to Solve Multiple Goals*, PhD thesis, University of Rochester, Department of Computer Science.

Lawrence, Steve; Giles, C. Lee and Bollacker, Kurt (1999), Digital Libraries and Autonomous Citation Indexing, *IEEE Computer*, 32(6):67-71.

Lin, Long-Ji (1993), Scaling up Reinforcement Learning for robot control, *Proceedings of the Tenth International Conference on Machine Learning*.

Maes, Pattie (1989), How To Do the Right Thing, *Connection Science* 1:291-323.

Maes, Pattie (1989a), The dynamics of action selection, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*.

Martin, Francisco J.; Plaza, Enric and Rodriguez-Aguilar, Juan A. (2000), An Infrastructure for Agent-Based Systems: an Interagent Approach, *International Journal of Intelligent Systems* 15(3):217-240.

McCallum, Andrew; Nigam, Kamal; Rennie, Jason and Seymore, Kristie (2000), Automating the Contruction of Internet Portals with Machine Learning, *Information Retrieval Journal* 3:127-163.

McFarland, David (1989), *Problems of Animal Behaviour*.

Minsky, Marvin (1986), *The Society of Mind*.

Minsky, Marvin (1991), Society of Mind: a response to four reviews, *Artificial Intelligence* 48:371-96.

Minsky, Marvin (1996), How Computer Science Will Change Our Lives, plenary talk, *Artificial Life V*.

Moore, Andrew W. (1990), *Efficient Memory-based Learning for Robot Control*, PhD thesis, University of Cambridge, Computer Laboratory.

Nilsson, Nils J. (1995), Eye on the Prize, *AI Magazine* 16(2):9-17, Summer 1995.

Noda, Itsuki; Matsubara, Hitoshi; Hiraki, Kazuo and Frank, Ian (1998), Soccer Server: A Tool for Research on Multiagent Systems, *Applied Artificial Intelligence* 12:233-50.

Numao, Masayuki (2000), Long-term learning in Global Intelligence, *17th Workshop on Machine Intelligence (MI-17)*, Bury St Edmunds, Suffolk.

Nwana, Hyacinth S. (1996), Software agents: an overview, *Knowledge Engineering Review*, 11(3).

Ono, Norihiko; Fukumoto, Kenji and Ikeda, Osamu (1996), Collective Behavior by Modular Reinforcement-Learning Animats, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*.

Paulos, Eric and Canny, John (1996), Delivering Real Reality to the World Wide Web via Telerobotics, *IEEE International Conference on Robotics and Automation (ICRA-96)*.

Pell, Barney (1993), *Strategy Generation and Evaluation for Meta-Game Playing*, PhD thesis, University of Cambridge, Computer Laboratory.

[PhD] is shorthand for [Humphrys, 1997].

Pollack, Jordan B. (1989), No Harm Intended: A Review of the "Perceptrons" expanded edition, *Journal of Mathematical Psychology*, 33(3):358-65.

Porter, Brad; Rangaswamy, Sudeep and Shalabi, Sami (undated), *Collaborative Intelligence - Agents over the Internet*, Undergraduate final year project, MIT Laboratory of Computer Science.

Ray, Thomas S. (1995), *A proposal to create a network-wide biodiversity reserve for digital organisms*, Technical Report TR-H-133, ATR Human Information Processing Research Laboratories, Japan.

Ring, Mark (1992), Two Methods for Hierarchy Learning in Reinforcement Environments, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB-92)*.

Rosenblatt, Julio K. (1995), DAMN: A Distributed Architecture for Mobile Navigation, *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*.

Rosenblatt, Julio K. and Thorpe, Charles E. (1995), Combining Multiple Goals in a Behavior-Based Architecture, *Proceedings of the 1995 International Conference on Intelligent Robots and Systems (IROS-95)*.

Russell, Peter (2000), *The Global Brain Awakens*, Element Books.

Simmons, Reid G.; Goodwin, Richard; Haigh, Karen Zita; Koenig, Sven; O'Sullivan, Joseph and Veloso, Manuela M. (1997), Xavier: Experience with a Layered Robot Architecture, ACM SIGART *Intelligence* magazine.

Sims, Karl (1994), Evolving 3D Morphology and Behavior by Competition, *Artificial Life IV Proceedings*.

Singh, Satinder P. (1992), Transfer of Learning by Composing Solutions of Elemental Sequential Tasks, *Machine Learning* 8:323-339.

Sloman, Aaron and Logan, Brian (1999), Building cognitively rich agents using the SIM_AGENT toolkit, *Communications of the ACM*, 43(2):71-7, March 1999.

Spolsky, Joel (2000), "Things You Should Never Do, Part I", Joel on Software, joel.editthispage.com

Steels, Luc and Kaplan, Frederic (1999), Bootstrapping Grounded Word Semantics, in Briscoe, Ted, ed., *Linguistic evolution through language acquisition: formal and computational models*, Cambridge University Press.

Steels, Luc (2000), The Emergence of Grammar in Communicating Autonomous Robotic Agents, *Proceedings of ECAI 2000*.

Stein, Matthew R. (1998), Painting on the World Wide Web: The PumaPaint Project, *IEEE / RSJ International Conference on Intelligent Robotic Systems (IROS-98)*.

Stone, Peter and Veloso, Manuela (2000), Multiagent Systems: A Survey from a Machine Learning Perspective, *Autonomous Robots*, 8(3), July 2000.

Sutton, Richard S. and Santamaria, Juan Carlos (undated), A Standard Interface for Reinforcement Learning Software, www-anw.cs.umass.edu/~rich/RLinterface/RLinterface.html

Taylor, Ken and Dalton, Barney (1997), Issues in Internet Telerobotics, *International Conference on Field and Service Robotics (FSR-97)*.

Tham, Chen K. and Prager, Richard W. (1994), A modular Q-learning architecture for manipulator task decomposition, *Proceedings of the Eleventh International Conference on Machine Learning*.

Tyrrell, Toby (1993), *Computational Mechanisms for Action Selection*, PhD thesis, University of Edinburgh, Centre for Cognitive Science.

Walshe, Ray (2001), The Origin of the Speeches: language evolution through collaborative reinforcement learning, submitted to the *6th European Conference on Artificial Life (ECAL-01)*.

Watkins, Christopher J.C.H. (1989), *Learning from delayed rewards*, PhD thesis, University of Cambridge, Psychology Department.

Weizenbaum, Joseph (1966), ELIZA - A computer program for the study of natural language communication between man and machine, *Communications of the ACM* 9:36-45.

Whitehead, Steven; Karlsson, Jonas and Tenenberg, Josh (1993), Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging, in Connell and Mahadevan, eds., *Robot Learning*, Kluwer Academic Publishers.

Wilson, Stewart W. (1990), The animat path to AI, *Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB-90)*.

Wixson, Lambert E. (1991), Scaling reinforcement learning techniques via modularity, *Proceedings of the Eighth International Conference on Machine Learning*.

Yahoo list of AI programs online, yahoo.com/Recreation/Games/Computer_Games/Internet_Games/Web_Games/Artificial_Intelligence

Yahoo list of ALife programs online, yahoo.com/Science/Artificial_Life/Online_Examples

Yahoo list of robots online, yahoo.com/Computers_and_Internet/Internet/Devices_Connected_to_the_Internet/Robots