3

Taking Experience to a Whole New Level

Luis Ignacio Lopera
Universidad de los Andes
Colombia

1. Introduction

Human kind through out history has shown a keen ability to learn by observation and to create. He's the only species on earth that has drastically changed his surroundings by constructing cities, houses and parks among other things. He also has left the planet for the nearest celestial body and built a home on the stars. But if one takes the knowledge needed to build something as complicated as the space station, one soon realizes that one did not have to learn everything at once. As matter of fact the knowledge needed to build the station is the result of a very long learning process that was done one step at the time.

This type of learning process, based very strongly on previous experiences, has proved to be efficient in the way that once something works it is fairly easy to replicate or do it better. However, it is interesting to point out that regardless if it is the best method for learning it is the only method used. The school systems all around the world expect a child to learn certain skills during the first years of schooling, such as reading, writing and spatial reasoning. Then these skills are broadly used from there on to learn things like basic algebra, logic reasoning, arts, crafts, history and so on. Once in college the student is expected to choose an area of interest and study the extra skills necessary to learn the advanced subjects of the area and be able to use them in a professional environment. If the student pursues a higher degree of education his success will reside on his ability to interconnect past experiences to produce some new bits of knowledge.

Interesting enough, the power of knowledge is derived not only from personal experience but from a collective experience as well. This can be seen in very isolated communities as well as in the global community of today. In aborigine tribes, the collective experience is passed from generation to generation usually by means of oral tradition. For example, the best way to hunt, the best grassing places for cattle and so on. Such knowledge is updated by the most recent personal experiences. In today's more globalized community experiences are shared through many different channels, such as books or the internet.

The discussion comes to the point where it becomes important to define experience. In a general context; the Merriam Webster's dictionary defines experience as:

"1 a: direct observation of or participation in events as a basis of knowledge b: the fact or state of having been affected by or gained knowledge through direct observation or participation

2 a: practical knowledge, skill, or practice derived from direct observation of or participation in events or in a particular activity b: the length of such participation <has 10 years' experience in the job>

Source: Machine Learning, Book edited by: Abdelhamid Mellouk and Abdennacer Chebira,
ISBN 978-3-902613-56-1, pp. 450, February 2009, I-Tech, Vienna, Austria

3 a: the conscious events that make up an individual life b: the events that make up the conscious past of a community or nation or humankind generally
4: something personally encountered, undergone, or lived through
5: the act or process of directly perceiving events or reality".
These definitions illustrate clearly how knowledge can derive from direct or indirect involvement in an activity. It also defines a way of learning. More precisely in the context of this book, "*Learning* is done by a machine when it records its experience into internal system changes that causes its behavior to be changed." (Looney, 1997). Most algorithms in machine learning use this definition to better adjust the detected classes and generate new ones if necessary.
But unfortunately, these inner changes do not take the machine closer to a human like learning method. It only perfects the machine output to a constrained set of variables. But if the set of variables, all of the sudden, become unconstrained or the constraints change drastically, the previews experiences become obsolete and the training process has to star all over again (Hagras et al., 1997). As a result, for machine learning applications, you want the problem to be as constrained as possible and the machine as invariable as possible. These limitations become the "*Achilles' heel*" of systems that have to undertake unexplored and unstructured environments.
Understanding human experience has been the material of study by many philosophers, and scientists. Is not the intention of this chapter to enter in the discussion on any way, however, it is relevant to point out that the basic definition given before falls short to describe experience that transcends the observed event's context; in other words, experience that is used in something else than the set of events where it was generated. This is best illustrated by an example: An electrical technician learned throughout his career how to repair CRT TV's, now he is faced with the challenge of repairing LCD screens. It is evident that some additional learning has to be done, but, a lot of the skills used to fix the CRT will be useful to fix the LCD. And furthermore, if another CRT TV comes to his shop, he would still be able to fix it.
From a systemic point of view, the agent's physical capabilities, such as sensors actuators, computational power, etc, can be considered *services* to the way of doing things. And these services become the framework to design and develop the architecture that will take experience to the next level.
The chapter starts the discussion by analyzing the way people carry out tasks, then introduces a concept of knowledge and its intricate relation to experience then a series of architectures are presented that illustrate the way next level experience can be implemented. These architectures are thought out to implement the ability, very often seen in human reasoning, of extrapolating experience; as in the example of the TV technician. The goal of the presented architectures is to establish the ways in which to use the agent's services to obtain the most of the agent's capabilities and increase the chance of success when faced with various problems and circumstances. Then it shows the application of one of the architectures to a theoretical problem and ends the discussion with some final remarks about the practical implications of using the proposed architectures.

2. Simplicity, fun facts of the way we do things.

'*STOP, think on what are you about to do!*', many times we have heard mothers instruct their children, usually because the youngster is about to harm himself, or engage in some mischievous behavior. This phrase is going to be the motto for this chapter's section.

Must people have certainly come across the annoying problem of having to fix a house appliance or an office gadget. And the resulting outcome for most of these people is to throw it away or call tech services. The focus of this section is the small portion of users who actually try to fix the broken object. For them here's the motto: *'STOP!!, think on what are you about to do!'*. Otherwise, how are we ever going to understand what's going on in the users' heads?

The problem of fixing things is very interesting to study the way we do things, mostly because it involves several brain actions/properties, like experience, analysis, observation, decision making, and coordination of movement, among others.

2.1 Case 1: opening the black box problem.

Once upon a time there was a black box. This box had a lid which was screwed shut with flat head screws, there where four of them one on each corner. The box had a "*broken*" behavior. (At this point it is irrelevant what the problem of the box is) To fix it, the person who's going to fix it (from here on, the fixer) must open the box and see what is causing the problem.

Inspired by the motto, an interesting question arises: What is the sequence of steps that are required to get in to the box? Let's follow a line of reasoning to find the answer to this question.

First step is always observation, observe the problem in detail and get as many characteristics as possible. From here the fixer will know things like there's a lid, there are 4 flat head screws, their position and size, and so on. It seems obvious after reading the problem's description, but bear in mind that the fixer is presented with the black box and not the description of the black box.

Next step is experience, the fixer must ask himself, "*Have I opened THIS black box before?*" if the answer to this question is YES, the answer to the "what is the sequence…" question is immediately found, the steps are somewhere in the fixer's head. But this trivial answer is not what we are looking for. If we take the NO answer, the following question arises: "*Have I opened SOMETHING LIKE this before?*" In this case a YES answer would lead to compare the black box, with every experience of opening THINGS LIKE this one, and using the best match to try and open it. In essence, finding similarities with previous elements would give us a starting point that is further ahead in the solution process than starting from scratch. On the other hand, the NO answer would lead to the next step.

Analysis, here the fixer must determine the type of tool he's going to use to unscrew the screws. Probably establish if there's a sequence to follow or just any random order will do the job, determine if it is sufficient to unfasten the screws or if they have to be removed completely.

The final step is action: the fixer does something, for example, removes a screw, from here on, the road can take two paths: trial and error or a methodic process of disassembly. Either one will get the job done, it's important to appreciate that in both roads the process becomes cyclic, as the fixer will have to stop and observe after each step is taken to determine if he's going to achieve his goal and apply this sequence of steps for each particular problem encountered. Figure 1, shows a flow chart of the Meta algorithm of opening the black box.

It is interesting to notice how there are two type of experiences that become very useful in this process. The first type of experience is very much like defined in section 1, and used widely in machine learning algorithms: direct experience over the event, the second type of

experience is an extrapolated experience, in other words, it is experience achieved in other events that is used to find a quick solution to the problem, a starting point further ahead in the road of solving the problem. As an example, opening the black box would allow the fixer to understand the way the computer's cover is quickly removed.

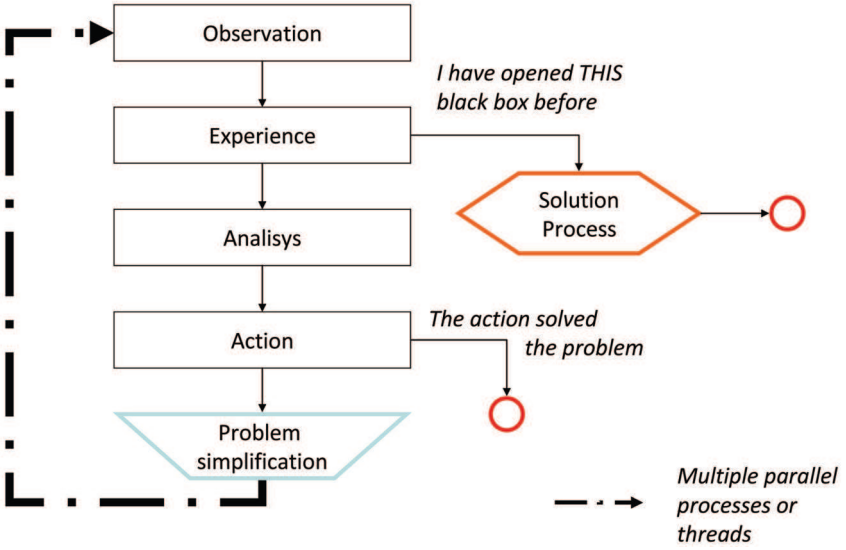


Fig. 1. Meta algorithm for problem solving

Other interesting observation on this case is the way it can be compared to recursive programming. In recursive programming the algorithm is called several times but every time with a simpler task, in terms, the same thing happens in case 1, the same four basic steps are recalled every time with partitions of the bigger problem.
Simplicity in this case is related to understanding that only 4 steps are needed, and that they repeat themselves over and over.

2.2 Case 2: fixing the black box's broken behavior.

At this point the fixer has opened the black box, and needs to fix the problem, as in the previous case, a very similar question arises: What is the sequence of steps needed to repair the problem? To find the answer to this question this time, we are going to take a different path; there is a meta-algorithm used widely for fixing things, it can be simplified to three stages as: Diagnose, repairing (replace, reposition, reconfigure, reinstall) and test.
With this meta-algorithm it is important to subdivide the task in two types. First type, it is the kind that comes with a manual, in this type of fixing, the fixer only needs to follow a set of steps designed to pinpoint the problem and fix it. Its only reasonable to mention that on this type of process, the fixer needs direct experience on how to solve the little details, the ones the "manual" assumes the fixer knows how to do. So, only one type of experience is needed. This is the type of activity people train for.
The second type of subdivision is the one with no "manual" or only limited information available. There are no steps or a determined sequence to follow, in this case (which is very interesting for this chapter), the fixer must use experience of different types to diagnose the

problem, and fix it. Interesting enough, if the meta-algorithm applied in case 1 is used for the diagnostics, repairing and testing stages, a solution to the problem can be found.

To illustrate, let's say the black box's broken behavior consists of a failure on an indication led that informs the status of the connection to a wireless network. Assume that the problem is a burnt resistor from the led's amplifying circuit. To understand what is going on (diagnostics stage) the fixer starts proving, looking to see if the box is actually able to connect, regardless of the *led*. The fixer must see that the box seems to work fine in this regard (*observation*), he turns to *analyze* the led's circuitry, un-solders the led (*action*) and tests it by itself. This because he knows from his *experience*, that L.E.Ds blow out rather often (It is important to mention that this is based on the fixer's experience, and only for the purpose of the example). When he finds that the led is not the problem, then he solders back the L.E.D, and starts checking for voltage level in the amplifying circuit until he finds the blown resistor. Again it is clear how the four basic steps of the meta-algorithm are used over and over again.

Then he gets the replacement resistor (repairing stage), un-solders the blown one and solders the new one. Repairing is usually a trained activity, therefore, this stage usually does not use the meta-algorithm; rather, it will use a list of steps or procedures. However, once in a while, to repair something the fixer must get creative. Assume now that he doesn't have the right value resistor, better yet, he has no resistors at all. He could run to the store and buy a new one; but again, not a very interesting solution. He could get the resistor from another broken gadget. In this case the meta-algorithm could be used to find and recover the part, and as it usually happens, the replacement is probably not a perfect fit, so he would have to use the meta-algorithm again to modify it and make it fit.

Finally testing, the fixer has to undergo a procedure to figure out if the repair was well done. Again we stumble with the duality of procedure vs. experience. The fixer could use procedures if they exist. But if not, he must rely heavily on experience to test the system until a suitable set of possibilities for failure is tried out and pass satisfactorily. In the example of the black box, it is rather simple: Activate wireless communication and see if the L.E.D blinks as it is supposed to.

With case 2 it becomes clear that there's a layer-like architecture to the process of fixing something. Upper layers determine the general procedure to follow, and lower layers take care of particular tasks. Furthermore, simplicity is associated to the use of the meta-algorithm in several occasions and contexts.

After having "stopped and thought" on what we where about to do to the black box; it is important to extrapolate at this point. If all possible problems are grouped together in to categories based on the agent's capacity to solve them, only three categories arise: problems which already have been solved, those which haven't and those which can't be solved by the agent. Those that have been solved become procedures like how to build a computer or a car, in the case of people, they could also become instinct, like running or dancing. Those that haven't been solved are the ones that present a challenge, and there's where the meta-algorithm comes in action, always observing, putting all other experiences to the test, analyzing and acting upon.

Although it is not the intention of this section either to undermine or to simplify the creative process, the act of problem solving of the human mind, which relies on creativity, can be approximated by understanding that a big part of the creative process comes from melding experiences achieved through out a series of events in a similar or even in completely different context than that of the problem at hand. A glance at the way any engineer's talent

evolves shows that although early stages could be magnificent, the best work is always later in the career because is fueled in part by the new experiences achieved in the early stages.

3. Storage, the key for knowledge.

Although the debate on a definition of Knowledge is still on-going, for all purposes of this chapter knowledge would be understood as defined in the Oxford English Dictionary: (i) expertise, and skills acquired by a person through experience or education; the theoretical or practical understanding of a subject, (ii) what is known in a particular field or in total; facts and information or (iii) awareness or familiarity gained by experience of a fact or situation.
It is interesting how knowledge and experience are intricately related. From the definition can be derived that since machine learning algorithms use a process of experience to better perform the given tasks, ergo, any system that uses a machine learning algorithm has *knowledge* of the specific task. The only problem with this statement is that by definition, knowledge seems to be a trait exclusive of a "person". Never the less it is still valid, if we understand a person as the ultimate system or agent. In other words, extrapolating the concept of knowledge to lesser systems, such as mechanical or electronic system, to describe the information, expertise and familiarity obtained through experience or education.
The term information is clear to see in current day technology, people store hundreds of thousands of information represented in bytes. It is also clear to see how a few fields in memory describing the algorithm's results or properties can be considered valid information, and that such can be acquired or refined through experience or programming (the equivalent of education in "lesser" systems),therefore also considered as knowledge. However, what to make of awareness and expertise? Can they be replicated in a non human system?
Expertise can be defined as the capacity of the system to carry out a task efficiently. Therefore, it can be replicated as it has been widely demonstrated that for certain tasks, machines are far more efficient than people. Awareness at a very primitive level has been replicated in machines (Bongard, Zykov, Lipson, 2006), and as a matter of fact is achieved through a method of experience. So, it is safe to extrapolate the term knowledge to a wider variety of systems.
A system has knowledge of how to carry out the task it is meant to do, because, in the worst case, the system was programmed to do it, since programming was proposed equivalent to education, the statement becomes true by definition.
But in the interest of this chapter, how does having knowledge take experience to the next level? From section 2 it can be determined that next level experience starts when the system can extrapolate what was learnt in one problem and use that to solve something else, and it ends when the system has evaluated the level of success on solving the problem. Then, knowledge of other problems is useful when using next level experience. But as experience goes up on level, so does knowledge, because by definition, if there is experience, the information achieved by it is knowledge.
A quick look on what could be seen in next level knowledge would throw probably some algorithms and some indicators on how efficient it was under certain circumstances. There would be an algorithm that would know how to choose and combine algorithms to solve new problems, and there must certainly be an algorithm that would store procedures that had effectively solved a problem. In this case, traditional machine learning algorithms and any algorithm designed to specifically solve a problem becomes an essential component to the algorithms found in next level knowledge.

People store information by creating interconnection between different neurons, part of that information, which is consider knowledge, is actually information about the way people carry out tasks. Some of it is fuzzy knowledge as the person knows that certain algorithm works well under certain cases in a certain way, while other may not work as well. There's also deterministic knowledge of this kind, for example, the way a person writes; clearly there is certainty that the algorithm for writing works every time.

Without the neurons' connections the storing of information wouldn't be possible, and without storage, comparison, characterization and choosing are not feasible. One of the reasons would be that there would be no knowledge (because there's no information) about the efficiency of an algorithm, so there would be no factors in which to base the choice other than randomness; there would also be no information to compare any two algorithms and no information about any algorithm could be generated because it would be immediately forgotten.

As in people, machines have various methods to store information. From the simpler latch or flip-flop all the way trough to quantum dots (Stick, Sterk, Monroe, 2007) and buckyballs(Anderson, 2007) passing by registers, and more traditional R.A.Ms, R.O.Ms, and magnetic hard drives. Although some neuroscientist despise the idea of comparing the human brain to a computer, some similarities can be pointed out; for instance, the "natural instinct" or "born instinct" can be compared to the functionality of the ROM in the computer, the short term memory to the RAM, and the long term memory to the hard drive.

Information in the brain seems to be stored in different sectors of the brain, depending of where it comes from or what it does; in a system, the information also has to be structure to achieve functionality.

By design artificial systems have a "natural" partition, in one hand there's the program memory while on the other is the data memory. In a way, this separates the "how to" from information, as mentioned before a program is knowledge achieved through "education" so this basic natural partition could be sufficient in some cases. However, the downside of this storage strategy is that the size of program memory is usually limited. This lack of space obligates to simplify algorithms and use only a small set of them. It also implies that the complexity of the higher level algorithms (HLA) is reduced to simple lookup tables as the actual algorithms could not be changed or manipulated.

In modern computing systems this lack of capacity is a matter of the past, today it is very inexpensive to have large amounts of memory available. This means a large number of programs and a large amount of information could be made available to a CPU or the processing unit of choice. Under this circumstances, HLA do not have to be limited to a look up table, they can be very sophisticated algorithm that could spawn new versions of basic level algorithms (BLA).

It is clear at this point that in order to have knowledge, there has to be a storage system. And such storage system has to be capable not only of storing data, but it has to be able to store algorithms as well, and if the HLA are sophisticated enough, it must allow them to manipulate the algorithms.

There are three characteristics intrinsic to a storage system of any kind. First of all it must have an appropriate capacity, not too much that the system would have trouble carrying the extra space not too little that algorithms could not work or be worked around easily. And second, the storage system has to be fast, even it means that it must compensate for latencies associated to slow media, it also means that it needs to be organized so it will find the data or algorithm that the HLA is looking for almost immediately. Last but actually the most

important, the storage system has to allow algorithm modification; with ever increasing complexity a good HLA could evolve an algorithm with time, so it is important to allow for such type of action over the algorithm.
Based on the second characteristics, the way an algorithm is stored has a great impact on the overall performance of the system. If the storage system is not fast enough, the system is going to have critical waiting periods while it loads the next algorithm to execute, and if such times are grater than the system's natural response time. The system could become unstable or collapse all together. Therefore it is crucial to structure the storage system to have a fast response.

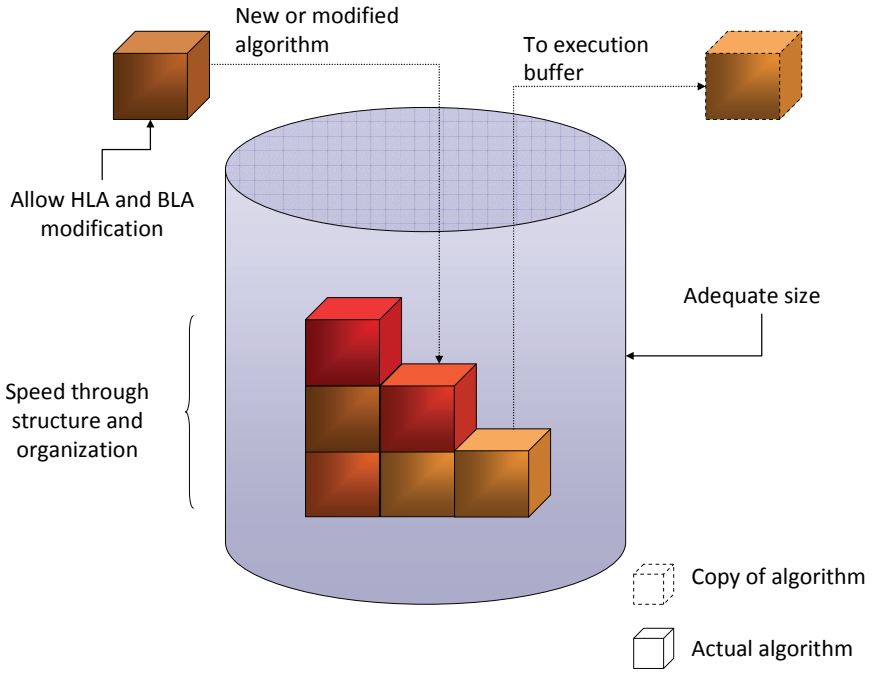


Fig. 2. Storage system characteristics

4. Architectures that allow for higher level algorithms.

Any means of storage could be considered a valid architecture for HLAs; however, it is important to keep in mind the three qualities associated for a good storage system for knowledge. Furthermore, any architecture has to provide the means to evaluate or at least have a grading mechanism to choose the appropriate algorithm for the given set of circumstances.
Without evaluation there's no experience to be achieved, because there wouldn't be the means to measure an improvement in certain task. In other words, if there is an HLA, it needs to keep track of how well it has resolved the problems at hand with the BLAs, meaning it needs to evaluate each BLA's performance. So whether the evaluation is

embedded into the HLA or its part of the system design and it is made available to the HLA as a service, it needs to be present.
Turning to the architectures, they can be divided into two groups, software architectures and hardware architectures. Although software architectures are the easiest to implement and the most familiar for developers, resent studies in hardware design are showing promising results.
Software based architectures have several advantages, for starters, must of the elements needed to create them are intrinsic to an operating system or a program i.e. multi-thread multi-process operations, file management or dynamic library loading. Other important advantage is the level of possible manipulation; an algorithm can be disassemble and assemble with changed properties. But the downside is that all that preparedness has a high cost in size, operating systems usually take a lot of space in order to give all that functionality as does the additional software.
In contrast, the speed achieved in hardware is dazzling, and with reconfigurable hardware techniques, drastically changing algorithms is possible. The problem is that there's a higher cost in design time, because all the interfaces needed to use massive storage, and reconfigure hardware have to be hard wired and hard coded; also there's less portability to other systems due to the hardware specificity.
Regardless of the technical issues that embrace each technology, it is important to take a look at some examples as for different practical problems there'll be a most appropriate implementation.

4.1 Software architectures: using a file system.

Despite the operating system of preference, it is going to present the developer with a file system. This File system allows the storage of massive amounts of information, and usually lets you handle multiple storage media like USB memories or hard drives with ease.
Figure 3 illustrates the basic layout of an architecture based on a file system. The evaluation subsystem could be an independent module; or as mentioned before, embedded in the HLA. The file type of choice is a dynamically linked library that can be loaded and unloaded as needed. The HLA is the entity that decides which algorithm to load based on the information stored in the evaluation file. The execution module runs the algorithm achieving a change on the system's stat; the efficiency and accuracy of the operation is measured by the HLA and the result is stored through the evaluation module.
The algorithms are recommended to be stored in compiled form, in other words in an executable format, i.e. *.dll* for Windows operating systems. This ensures a faster execution and allows the direct interaction with all of the systems' services; also it allows the direct use multi thread technology, leaving the responsibility of processor time assignment to the operating system.
Interpreted formats, like a Matlab file, are not recommended for the BLA as they become costly to execute because they have to load the interpreter. Also the algorithm has to use the interface provided by the interpreter in order to access the system's services, this usually has an impact on performance and some services are restricted. Things like multi threading depend exclusively on the interpreter of choice so it is not always available. In (Lopera, 2005) the Matlab algorithms always caused the execution time to default to the worst case.
When regarding direct algorithm manipulation by the HLA, a few things have to be taken care of. First of all naming new libraries, the HLA has to keep track of the new libraries created otherwise it might not keep an appropriate performance log and thus, it might not use the newly created algorithms even if they turn out to be more efficient.

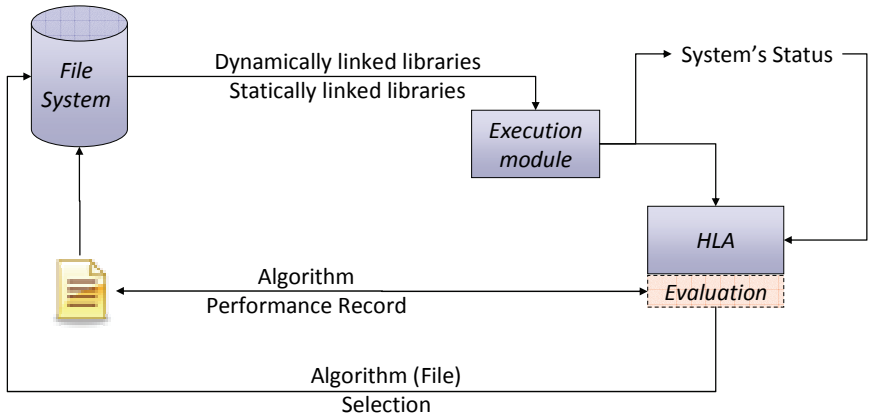


Fig. 3. Basic layout of file system based architecture

Algorithm manipulation is easier to do in interpreted formats because is a natural way to partition and mix functionalities, in compiled formats, it requires more steps but it can be done, weather is combining at a source file level and recompiling, or mixing in binary format; which it hasn't been tested and requires a profound knowledge of the binary structure of the compiled library. This also means that the HLA has to keep track of what source code belongs to which library.

One of the advantages of file system based architecture, especially when using compiled format, is that the system will only load what ever algorithm is executing and the system's services, nothing else, so it can be very efficient in respect to memory usage.

By designed, a file system complies with the characteristics proposed for knowledge storage; however is the responsibility of the HLA to keep the order and structure of the file system. A poorly designed HLA can end up clogging the file system surrendering it inefficient and ultimately halting the system. Other advantage of the file system is its portability; the hardware architecture is some what transparent, as long as it supports the operating system: it will support the file system.

The disadvantages lie with the evaluation module; because is a file based module, all searches have to be carried on within files, so a lot of searching and updating functions have to be written in order to allow the HLA to effectively evaluate and choose BLAs.

4.2 Software architectures: databases

The database architecture is an expansion of the file system architecture; it seeks to improve where the file system presents its most weaknesses. It also takes care of the evaluation structure, which allows having multiple HLAs that share the same information about the algorithms and simplifies overall HLA development.

A database is designed to store information, and as such it allows storage of multiple types of information in an orderly fashion; its internal structure is designed to relate information between tables so it facilitates data management and storage structure, furthermore, it also specializes on information retrieval; it is designed to fetch huge amounts of information in short periods of time. This makes it ideal to take care of storing the algorithms in binary

form as well as in source code form, associating all sorts of parameters that allow the HLA to choose the best algorithm for a more complex context.
Figure 4 illustrates a general architecture, in this case the HLA works with the database to manipulate and evaluate the stored algorithms, once it has chosen one, it retrieves it and saves it to the file system for execution. This because most operating systems don't allow executing information that is considered data, except for executable files on the file system.

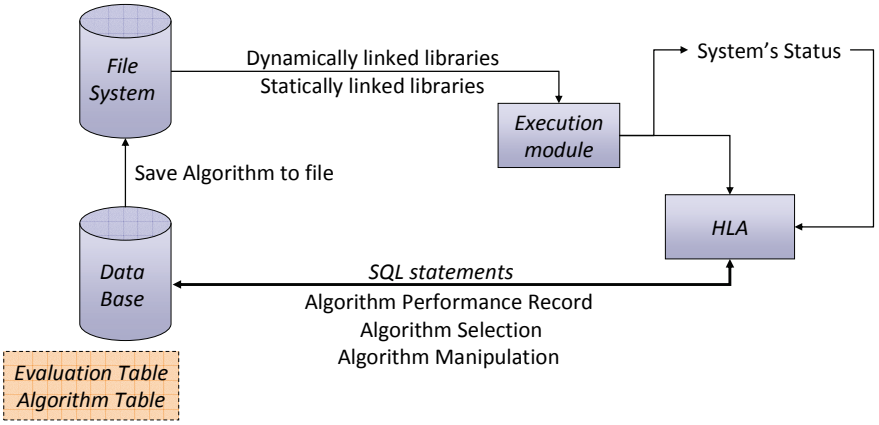


Fig. 4. Basic layout for the database architecture

As mentioned before, the database improves performance and facilitates the job of the HLAs, at the cost of having to load the database server which implies some memory usage and processor time; however for most systems based on pc computers this is not a problem. The advantages outweigh the cost. In (Lopera, 2007) there is an interesting analysis about the pros and cons between both architectures.

4.3 Software architecture: when space is limited

This type of architecture is considering systems that are developed using microcontrollers where access to memory resources is limited and no operating system is available or does not have file system capabilities much less a database server.

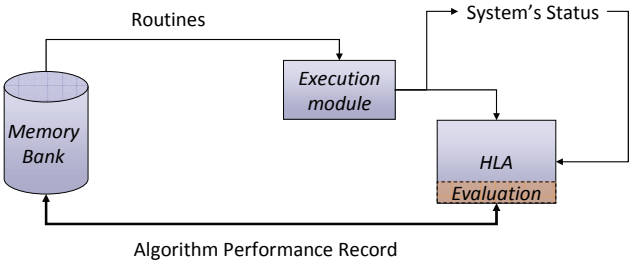


Fig. 5. basic layout for limited space architectures

In this case HLA must have embedded the evaluation module; it should work over a memory area, keeping a rather simple record of performance and link to the respective program counter's position of each routine. It is recommended that the routines be constructed in an interrupt basis so in that way they'll return handle to the HLA so it will be able to monitor the system status and the routine's performance.
To achieve some level of routines manipulation they should be parameterize, in that way the HLA can modify the parameters to fine tune the routine's efficiency.

4.4 Hardware architectures: reconfigurable hardware

A typical architecture for reconfigurable hardware is the cooperation of a processor unit with a programmable electronic device (PED) as PSOC or FPGA. In this configuration the processor has the responsibility of programming the PED, and for that, the processor uses storage memory to store the binary files that contain the programming sequences, usually downloaded in to the PED through JTAG.

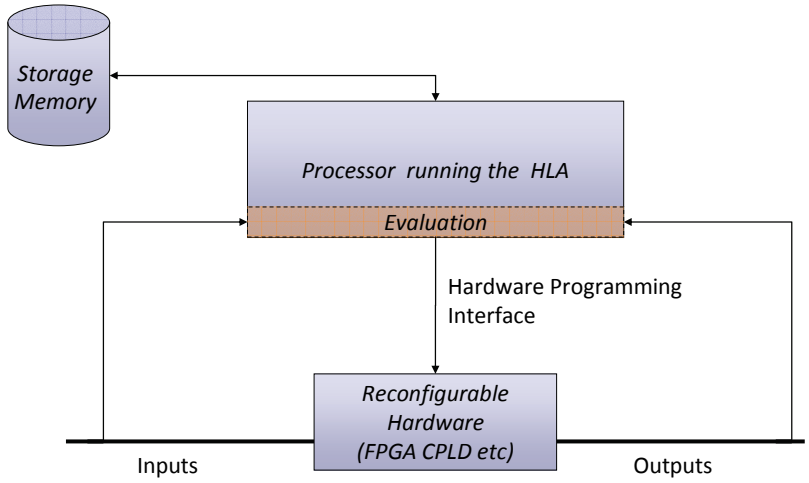


Fig. 6. basic layout for reconfigurable hardware architectures

In this case there are several configurations that can be carried out, and they all depend on the capacity and speed of the processor as well as the PED. For instance the evaluation module can be run at the processor along with the HLA or can be programmed and configured in the PED so it will match its internal configuration and facilitate performance measurement.
The HLA is recommended to be executing in the non-reconfigurable part of the system as it is pointless to load and reload every time the PED has to go through a programming. This takes up some time, and could compromise system's stability.
Some of this configurations support small operating systems, this operating systems could run small database servers, in this case leaving the BLA to be implemented at hardware level. This certainly has some performance issues that have to be evaluated based on each specific application.

One of the advantages of this architecture is that PED have become interestingly complex and powerful as they have grown in capacity, mixing microcontrollers with analog cells and digital cells. This resource availability can be used to implement high performance BLAs using very up to date design techniques.

4.5 Hardware architectures: non reconfigurable hardware

Not all types of algorithms are worth the trouble of implementing at a hardware level. In most cases due to the repetitiveness and the sequential nature of its internal operations a software architecture is more suitable. Even though, parallel processing, state machines, and other hardware design techniques can be embraced to implement powerful solutions.

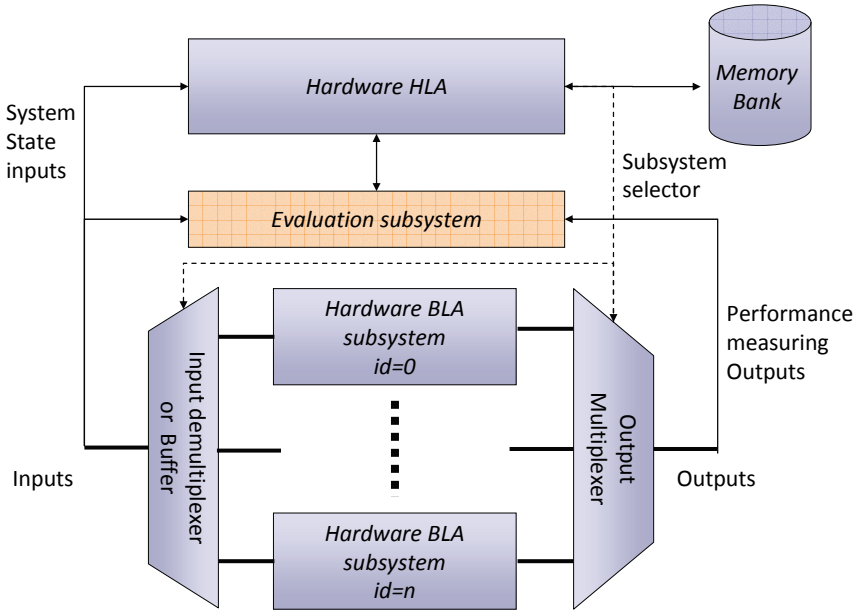


Fig. 7. basic layout of non reconfigurable hardware architectures

The way this architecture works is as follows: The HLA controls the output multiplexer and input buffer (or demux) it also must enable the chosen subsystem that will operate over the inputs and produce the appropriate outputs. This choice is based on the information stored in the memory bank. The evaluation subsystem is constantly monitoring the system's state inputs and the outputs selected to measure the hardware algorithm's (HA) performance; it also communicates the results to the HLA which in turn stores that information into the memory bank.

The hardware HLA, from the hardware design point of view, could be conceived as the control unit of the system. The evaluation is considered a separate module in this architecture because based in good hardware design strategies modularity is enforced and

since its job is so distinctly clear and does not mix with any other process the HLA might be doing. The input buffer has to be designed so it will present in adequate form the inputs to the HBLAs, just wiring every system input to the HBLA's input might encounter fan-in, fan-out or loading issues. The output multiplexer is pretty straight forward, the only concern is the signal types, in which case, an appropriate multiplexer has to be designed.
Unfortunately this architecture is the most expensive to implement and the most keen to present problems do to implementation, i.e. wiring and signal coupling issues. Despite its cost and arduous construction, it is worth while presenting this architecture as it illustrates how the HLA can be taken to the must basic level. It also reinforces the following concept: the importance of basic level modularity, which is going to be presented in more detail in section 5. In other words, regarding HBLAs inputs and outputs, they all have to talk the same languages, since they will be connected to the same interfaces; this becomes an important design restriction.

4.6 Remarks

The presented architectures show some alternatives of how to implement HLA and the evaluation mechanism, which are necessary for higher level experience. In general, Figure 8 shows a basic hierarchical structure of how to design an architecture that is considered HLA enabled.

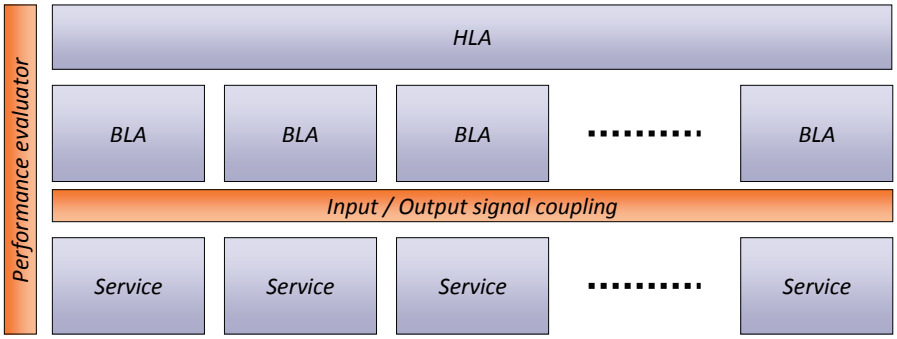


Fig. 8. general architecture

Service based design is crucial for these types of architectures. This way, each BLA knows exactly how to talk and listen to a system service. It also allows the execution of multiple BLA that use independent services at any one time and simplifies the design of the BLA as it only needs to interface with services it requires.

5. How to design robots with Higher Level Algorithms

This section analyses the design procedure of a mobile robot, it does not design a robot itself but assumes that there is certain mechanical infrastructure, hardware, and even software at a service level. The center idea is to structure the general architecture at a high level. For this we assume that the robot is at an advanced stage, in other words the first elements of the design process have been taken care of, the basic physical structure, the control system and electronics of the individual elements like motors, arms, cameras, etc are up and running.

5.1 The robot

BoBoT, which is going to be the robot's name, has three main service subsystems: the first one consist of a set of four independently driven wheels; second, 2 grippers each mounted on a arm with 2 sections and 2 degrees of freedom for each joint (3 in total); and third it has a bundled dual camera system with pan, tilt and zoom capabilities. The brain of the operation is going to be a laptop system and the database architecture is going to be used.
BoBoT is also equipped with a series of sensors that complement the basic instrumentation used to achieve control of the service subsystems:

- A three axis accelerometer
- An up down sensor.
- An applied force sensor for the arms.
- A battery charge meter.
- A GPS

Figure 9, Figure 10 and Figure 11 show the black box models of the service subsystems

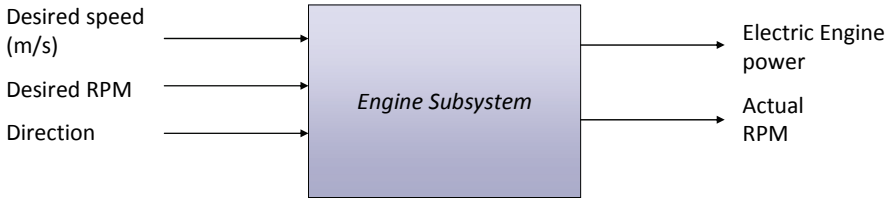


Fig. 9. Black box diagram of the engine subsystem

The engine subsystem has two ways of operation: it can turn by specifying an actual desired speed in meter per second; the engine will turn in the direction implied by the speed's sign. The other way is to establishing the RPMs and a direction. To specify which input to listen to, the unwanted one has to be set to 0. If not, desired speed prevails. In turn, the engine subsystem's outputs inform of the power given to the motor and the measured RPM s.

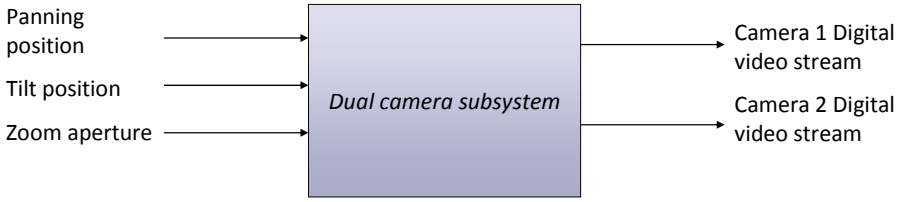


Fig. 10. Black box diagram of de dual camera subsystem

To operate the dual camera subsystem it is sufficient to specify the position in the pan and tilt axis, and how much zoom is desired, the cameras can not be controlled separately. The outputs are the two video streams in a mildly compressed digital format.
To use the arm it is important to understand that the grip operation is independent of arm operation. The grip has two ways of operation: one is by establishing a desired action and a speed of the action, i.e. "close" "fast" and the other is by establishing the action and the forced to be applied, i.e. "close" "hard". In the first example, the grip will close fast and

apply maximum force, in the second it will close slowly until it reaches the desired applied force. The system will constantly give out the grip status, i.e. opened, opening, closed, closing, and the actual force applied.

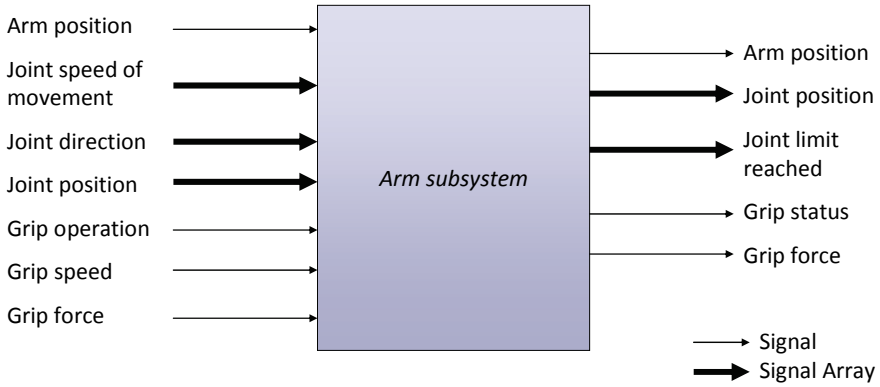


Fig. 11. Black box diagram of arm subsystem

The arm can be operated in three different ways: In the first one, the grip can be positioned in a 3D space with origin at the shoulder. The second way, allows positioning each joint accordingly. And at last, a joint speed and direction can be specified in order to achieve constant movement. And as outputs there are: the grips position with respect to the shoulder, joint position in their local coordinates, and an indicator if any of the joint's limit sensors was reached.
BoBoT has two arm subsystems, 4 engine subsystems and 1 dual camera subsystem.

5.2 The things BoBoT can do:

As part of the design process it is important to know precisely what it is expected of the robot. This section assumes that the robot has to carry out the following actions:

- Vision based navigation with global positioning
- Vision based navigation with inertial positioning
- Vision based navigation with visual terrain recognition for positioning
- Wide turns, forward and backwards.
- Rotations around wheel base center
- Pick up and place delicate objects.
- Pick up and place sturdy object.
- Variable speed and direction.
- Movement with the arms
- Swing
- …

These actions also show that there are commonalities between them, and also give the sense that there is more ways to achieve success, or that they share a common goal, i.e. the first three, the ones using vision based navigation, share the goal of moving from one point to another.

The next step is to identify the possible BLAs, as mentioned before the BLAs have to be extremely modular, so the expected actions not necessarily become BLAs. For instance, to pick up an object BoBoT will have to use vision to identify the object's position and use that position to place the grip at a gripping distance, regardless if it is delicate or sturdy. Thus, there are at least three BLAs, one for object location, one for arm movement, and one to identify if the object is delicate or not so BoBoT can actually grab it.
There can be multiple versions of the BLAs, in the picking up example, moving the arm could be done by controlling the trajectory in a 3D space assuming the trajectory is clear, or also assuming a clear trajectory but monitor the arm's applied force sensor to detect collision, or use the cameras to check for obstacles. If used the latter, the importance of modular service design is critical as the camera would be used by two BLAs. When using HLAs, there's no need to choose one of these three approaches to the same problem, instead you can store all three BLAs and have the evaluation subsystem evaluate them under different circumstances.
To further reassure the importance of modular service design, at least three BLAs can be designed to use the arm modules, one for each input pair, one for 3d positioning, one that uses joint positioning, and other one that uses joint movement. In this case it is simple to develop the BLA, but if instead there were no good service design, each BLA would have to deal with problems related to the direct control of the arm, and maybe wouldn't be as easily interchangeable or their size and complexity would increase.
Once identified all the BLAs with their different versions, the next step is to write them, compile them and individually test them. Also the BLAs have to be tested in group as the way they are expected to be used and correct any interfacing problem that might result from things like resource sharing.

5.3 The storage strategy

Having tested all the BLAs, it is needed to gather the following information:

- Excluding BLAs, those that perform different tasks but can not run at the same time.
- BLAs that perform the same task but in different versions
- Qualifiers of BLA performance
- Environment status variables in which each BLA out performs the others in the same task.
- BLA parameters if any.
- BLAs needed to perform each action
- Qualifiers of action's performance; how efficient was BoBoT to perform the task.
- Switching task times, it is easier to manage system stability at a HLA level, but it only matters when switching times are really critical.
- Which subsystems are used by each BLA

If at this point some incongruence is found among the BLAs they must be corrected before continuing because they might induce critical changes that force to repeat the previews steps.
With this information the database tables can be created; it is recommended but not strictly necessary to: 1 BLA table, 1 BLA parameters table, 1BLA evaluation table, 1 action table, 1 action evaluation table. For the action table it is recommended to use a code, if space is sufficient an extra table could be used to store that code, but it would only be useful for the

developer or generating reports, it wouldn't have any effect on the HLA. Figure 12 shows a possible table setup.

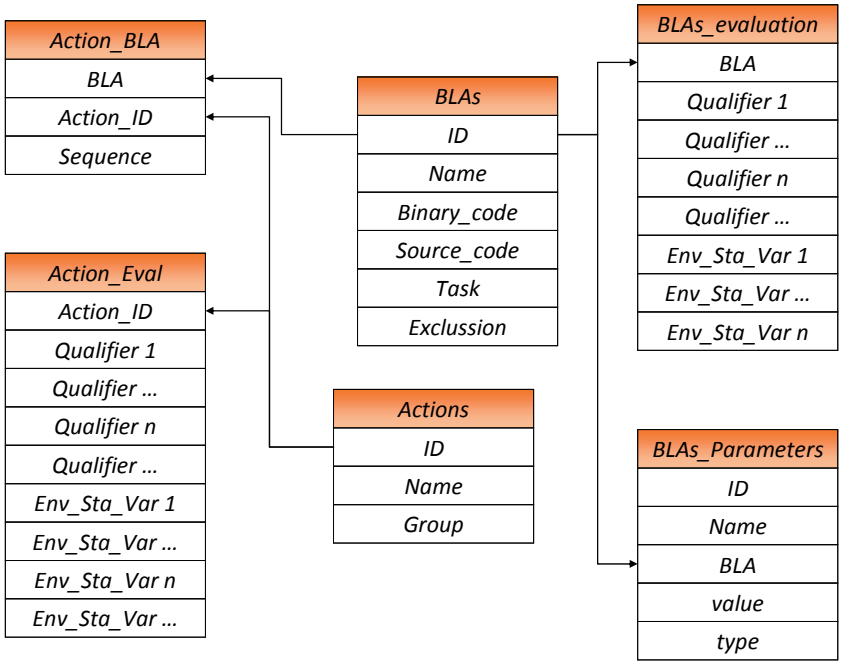


Fig. 12. Table reference diagram

In this setup, there's the *Actions* table that stores the coding but the additional field of *group* allows identifying which actions are the same, so they can be evaluated and associate different, but equivalent, BLAs. Also the *Actions_Eval* table stores information about the Environment Status variables so the HLA can track which combination of BLAs worked best for those conditions of the environment.

In case of using other HLA architecture, the same steps can be followed, only the storage structuring has to be adequate to the choice.

5.6 Finally the HLA

The HLA could have several roles in BoBoT, it could be in charge of fulfilling a mission, deciding the best way to successfully complete it. In this role the HLA would work with the *Action* tables evaluating and calculating constantly course of action, and how far it is to completion.

Other role the HLA could assume is to take a course of action from a user, and follow it; in this case the HLA would work closely with the *BLA_Evaluation* table to choose the best BLA for the given conditions and course of action. A course of action can be expressed in terms of the *Actions* table, the corresponding BLA retrieved from the *Action_BLA* table and the best

BLA from the *BLA_evaluation* searching among the algorithms that share the same value in the task field and are none excluding.
It is important to keep clear the role the HLA is going to take and how is going to take advantage of the tables, if several roles are detected it is a clear sign that the HLA has to be broken into modules, one for each role, and each module assume the appropriate hierarchy. If it turns out that there's something on top of the HLA, those on top could be considered next level HLAs.
Once the HLA's role is established, the type has to be chosen, and there are two types HLA: Those that have programmatic responses, and those that have learnt responses. HLAs with programmatic responses are those algorithms that have transfer function or some mathematical equation that relates the inputs to the outputs and are programmed. In the second type, the HLA learns from experience, it tries actions, evaluates performance and start to mix accordingly to achieve better results. Thus, this type of HLA could be any of several machine learning algorithms, working with other algorithms and a sub set of inputs.
Into what BoBoT is concerned, BLA of al sorts could be written, i.e. to use the four engines individually, in pairs or all together, to use each hand separately or gracefully coordinated, visually inspect the world surrounding him and use vision for a diversity of tasks. He 'would be able to successfully complete hundreds of mission of all sorts.
The level of success can be associated to the complexity or smartness of the HLA, for instance, a very programmatic HLA that was designed for a very specific and stable environment would certainly fail on dynamic environments. However, an adaptive HLA that takes record of how the environment affects its BLA's performance is more likely to succeed.
One of the advantages of using HLAs is that they force the design to be so modular that new BLA could be introduced and the previous work wouldn't be wasted, it will let the HLA evaluate and choose and optimize procedures, and user machine interfacing is done at a more natural way since it could be done by describing actions.
The storage strategy is open for the designer to best choose the tables or structures he needs, and allows to be as sophisticated as to have several levels of associations, or as simple as a few register in the memory bank of a microcontroller.

6. Being practical, final remarks.

In this chapter the discussion has focused on the how to and the what, but it is important to reflect on the "if we should" or the "is it worth it".
A NASA rover sent to mars, even though it seems a promising scenario, is not the best candidate for HLA, at the first glance, because putting it on mars cost a lot of money; and just to have it start trying stuff that won't work and that might cause an unpredictable failure it would be too risky. However, if once the rover has acquired the relevant information materials pictures etc. putting it to try out BLA becomes interesting, at least more interesting than letting it rot there.
The horse gait problem proposed in (Lopera, 2007) which is actually an energy optimization problem is a good example of the power of modularity since each leg is driven differently on each gate, but is worth the trouble of installing an HLA? There's a trick to this problem, and that is that depending on the terrain, especially on its slope, the gait has to be modified

drastically thus its energy consumption. Furthermore, if the gait algorithm (BLA) can be parameterized in order to adjust leg position and rhythm, the HLA becomes a powerful tool since it will start evaluating and adjusting those parameters so the horse would be able to keep doing the gate. But as far as the optimization problem, there would be the need to generate an additional level in which to operate in terms of the speed achieved by each gate, the energy it consumes and the track's layout.

In an industrial application, there's no need to have HLA, because once the process is optimized it would operate like such. The process sequence is usually determined by its nature and there are optimization techniques and algorithms that do this type of process fine tuning rather well.

In multiprocessor/multicore architectures HLAs could be used to supervise the execution of several learning algorithms in parallel to find optimums in highly complex functions. Since it can analyze the topology of the function, it could use the best optimization algorithm for the area of search. In that way, it could also be used to automatically evaluate classifying algorithms.

The appropriate scenarios for HLA are those that present high environment variability, or are highly unstructured, have several possible BLAs, and there's good computational power and memory availability.

The use of HLAs serves as implementation to the problem presented by (Van de Velde 1995) as to how internalize representations. As he puts it in his child example, the walk by holding a hand is an infant BLA that the HLA will perfect until it has a walking by own means BLA, thus constituting the internalization.

7. Future research

There's an interesting discussion, which this chapter purposely avoided getting in to, about if these architectures could be considered as epistemological. It would be interesting to compare what experts in this area have to say.

One line of research that emerges naturally from this proposed architectures, is the involvement of other natural concepts that participate in the experience process, for instance: What use would have concepts like pain or tiredness for a system that has the capacity to choose the way to solve a problem? How could they be implemented and interconnected with the presented architectures?

The presented architectures have a strong hardware based, reality measuring and affecting feeling to it, since they where thought out for physical systems as mobile robots and such. However, it would be interesting to measure the effect of the architecture in purely virtual systems. How would it affect performance compared to more traditional implementations?

And the last couple of question that emerges from this line of reasoning are: How to code creativity? And would we be able to create a HLA that has creativity as one of it biggest traits?

8. Conclusion

This chapter described a few architectures that support a higher level of experience; however they are not the only architectures possible. Any architecture that evaluates and

records the performance of basic modules and uses that information to decide which module to use or how to adjust the module's parameters is considered an architecture that supports higher level experience.

It is clear that the intricate relationship between knowledge and experience can be constructed on an artificial system. Furthermore, it can be generated by the system if its architecture and available resources allow it. Unfortunately, the power of the relationship between knowledge and experience and how the system embraces that power is only as good as the HLA allows it to be. In other words, a lookup table HLA would never be able to undertake tasks for which the environment parameters are not within the lookup table.

The architecture has to be carefully chosen for the resources available and the complexity level of the system. As mentioned before, their use in invariant environments, invariant systems and where no learning is involved, becomes a waste of resource and could compromise development time. But, in the other hand, there is little or no knowledge about the environment and it is desired to maximize mission scope, then architectures that support next level experience could simplify the problem dramatically. This simplification occurs in part because the designers do not have to resolve all the possible problems the system could encounter. Instead they solve basic issues, and leave problem solving to the system.

This type of architecture meets the definition by (Van de Velde 1995) of intelligent systems. As it has cognitive knowledge of its environment as evaluation criteria for the BLAs obtained through the inputs subsystems, and uses that knowledge to determine appropriate course of action, establishing a behavior in its environment.

9. References

Josh Bongard, Victor Zykov, Hod Lipson, (2006) "Resilient machines through Continuous self-modeling", Science 17 November 2006: Vol. 314. no. 5802, pp. 1118 – 1121, DOI: 10.1126/science.1133687

Hani Hagras, Martin Colley, Victor Callaghan, (2001) "Life Long Learning and Adaptation for Embedded agents operating in unstructured Environments", IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th Volume 3, Page(s):1547 - 1552 vol.3.

Carl G. Looney (1997), Pattern Recognition Using Neural Networks, oxford university press.

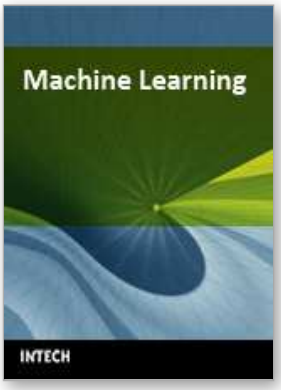
Luis I. Lopera, (2005) "S.N.A.P.A. 'Supervision, navigation and planning architecture': arquitectura de navegación, planificación y navegación para un dirigible no tripulado, Tesis de maestría, Universidad de los Andes, Bogotá Colombia, 2005.

Lopera, L.I.; (2007) "Algorithms Storage System", Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007 25-28 Sept. 2007 Page(s):370 – 375 Digital Object Identifier 10.1109/CERMA.2007.4367715

Anderson M, (2008) "Buckyballs to boost flash memory", IEEE Spectrum, June 2008, Page 15

Daniel Stick, Jonathan D. Sterk, and Christopher Monroe, (2007) "The trap technique toward a chip based quantum computer", IEEE Spectrum ONLINE, First Published August 2007.

Van de Velde W, (1995) "Cognitive Architectures - From Knowledge Level To Structural Coupling", L. Steelss (Ed.) The biology and technology of intelligent Autonomous Agents. NATO ASI Series, Series F: Computer and systems Sciences, Vol. 144, pp. 197-221. Springer, Berlin



Machine Learning

Edited by Abdelhamid Mellouk and Abdennacer Chebira

ISBN 978-953-7619-56-1
Hard cover, 450 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Machine Learning can be defined in various ways related to a scientific domain concerned with the design and development of theoretical and implementation tools that allow building systems with some Human Like intelligent behavior. Machine learning addresses more specifically the ability to improve automatically through experience.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Luis Ignacio Lopera (2009). Taking Experience to a Whole New Level, Machine Learning, Abdelhamid Mellouk and Abdennacer Chebira (Ed.), ISBN: 978-953-7619-56-1, InTech, Available from: http://www.intechopen.com/books/machine_learning/taking_experience_to_a_whole_new_level

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821