Distributed Artificial Intelligence: A Team-Based Approach to Cognitive Architectures

Lucas Cook
Department of Computer Science
Western Kentucky University
Bowling Green, KY 42104
lucas.cook238@topper.wku.edu

***Abstract*—Distributed Artificial intelligene (DAI) is the study of distributed problem solving across multi-agent systems (MAS). With the field of artificial intelligence (AI) exponentially growing and entering our everyday lives, it is important to know how we can optimize its performance. Through the use of multiple intelligent agents, DAI is able to improve on the centralized AI system in terms of efficiency, reliability, scalability, cost, and possibly much more. I will be discussing what makes up an agent, how they cooperate in multi-agent systems, and construct a basis of the benefits and current complications of DAI.**

Keywords: distributed artificial intelligence, multi-agent systems, agents, cognitive architecture, reinforcement learning, homogeneous, heterogeneous, consensus, omniscience

I. Introduction

Distributed computing is when several computers can communicate between each other, often over a network, and perform software functions as one. Now imagine if you make each computer intelligent using what we call agents. The possibilities of what one agent can accomplish in centralized artificial intelligence (AI) is facinating on its own, but what if we combine agents using distributed computing? This is called a multi-agent system (MAS), which is used in the field of distributed artificial intelligence (DAI), and is, in my opinion, the future of not only AI but technology as we know it. Throughout this paper I will be discussing the differences in agent types, the structure of cognitive architectures, characteristics of MASs, challenging issues with MASs, real-world applications, and the future of DAI.

II. Introduction to Rational Agents

According to Russel and Norvig, an agent is an autonomous entity which observes through sensors and acts upon an environment using actuators, and that AI is viewed as the study and construction of rational agents [1]. An agent's sensor can be anything from a camera to an ultrasonic sensor, and is also known as the input device for environmental data. An agent's actualtor is its method of output, such as wheels, a robotic arm, speech, etc. The environment that an agent works over can be the real world, a subset of the real world, the internet, a collection of other agents and their states, or possibly a combination of these [2]. Some agents have identical design but different roles and/or capabilities, this classifies them as heterogeneous. Heterogeneosity is what allows multi-agent systems to thrive bettering its perception decision making, and in a single-agent system used in centralized AI, this quality is simply nonexistent [3]. Agents that contain identical knowledge and action sets are called homogeneous and are good for increasing the efficiency of cooperation. Agents can be categorized into five classes based on the level of intellectual capabilities. These classes include simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents, and learning agents [1].

A. Simple Reflex Agents

Simple reflex agents are the most basic of intelligent agents. They simply analyze the environment them through a sensor and make an action iff it meets a defined condition in its static condition library. This type of agent does not store the current state of the environment, it simply just acts upon what it observes.

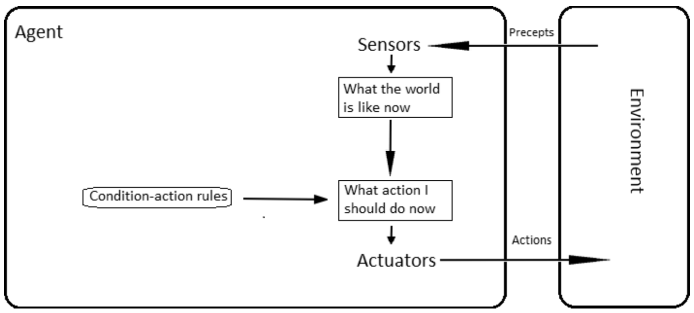

Fig. 1: Simple Reflex Agent Architecture [1]

Russel and Norvig like to use the analogy of putting your hand up to a flame to describe its intellectual process. The human reflex when putting their hand up to a flame is to pull away, and is an example of a condition-action rule [1]. The condition-action rule for this occurance would be:

if hand is in fire **then** pull away hand

Due to the referencing of a static condition-action rule library, the simple reflex agents only base their decision on their current perception of the world and not any past perceptions. One known problem of simple reflex agents is that they can get stuck in an infinite loop if the sensors recognize a condition in its library and repeatedly call on its action. By observing Figure 1, it is evident how an infinite loop can occur.

B. Model-Based Reflex Agents

Model-based reflex agents have the same condition-action process as a simple reflex agent, but they are able to handle environments that are partially observable or accessible. They accomplish this by storing the current state of the world, and remembering all past states of the world within its long term memory. This allows the agent to describe parts of the world that cannot be seen from the current state, allowing them to make decisions based on their percept history. Russell and Novig claim that in order to update the internal state it requires two things: Information on how the world evolves on its own, and how the world is affected by the agents actions [1]. This knowledge of how to world works on its own and with agent interference is called the model of the world. Figure 2 displays the architecture of a Model-Based Reflex Agent and how the state update occurs.

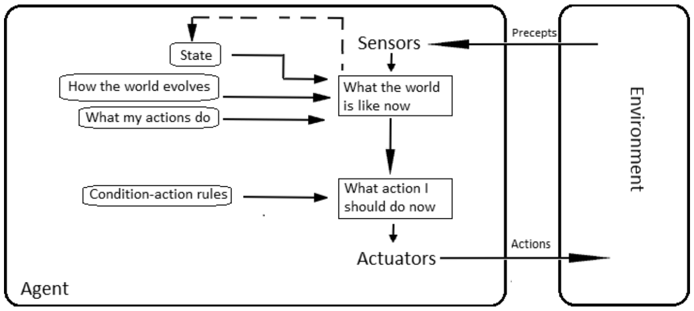

Fig. 2: Model-Based Reflex Agent Architecture [1]

C. Goal-Based Agents

Goal-based agents are model-based reflex agents with added internal expectations. Goals are a set of desired situations and by adding the goal component to a model-based agent it creates the first instance of AI search and planning. Unlike reflex agents, goal-based agents do not have automated responses to conditions. Instead, goal-based agents review over all of the possible responses tothe current state (search) and choose the one that puts them closer to their goal (planning). This concept is the origin to Reinforcement Learning and will be further elaborated on later in the paper. As a result of this search feature, goal-based agents are said to be more flexible than model-based reflex agents due to their ability to update their own actions as opposed to having to re-write and re-code the condition-action rules. For example, if a mars rover needed to get up a hill, the agent can update its knowledge on how much power to put into the wheels to gain certain speeds, the agent will store this new knowledge within its long term memory and all similar behaviors will now automatically follow the new knowledge on moving based on this state [1]. Figure 3 displays the Goal-Based Agent archiecture and how search and planning concepts are implemented to achieve the goal.

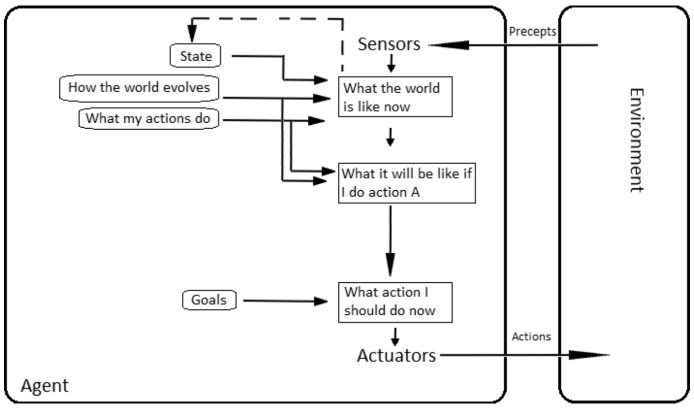

Fig. 3: Goal-Based Agent Architecture [1]

D. Utility-Based Agents

Utility-based agents are very similar to goal-based agents but with one main difference, numeric efficiency. Goal-based agents process actions internally to see if their goal is achievable after the action, then follow through with that action if it leads the agent closer to its goal. What it does not account for is if there are several ways to accomplish the goal. This is what the utility function does on a utility-based agent. Figure 4 shows the Utility-Based Agent's architecture and its implementation of the utility function in its decision making process. The utility function processes all of the

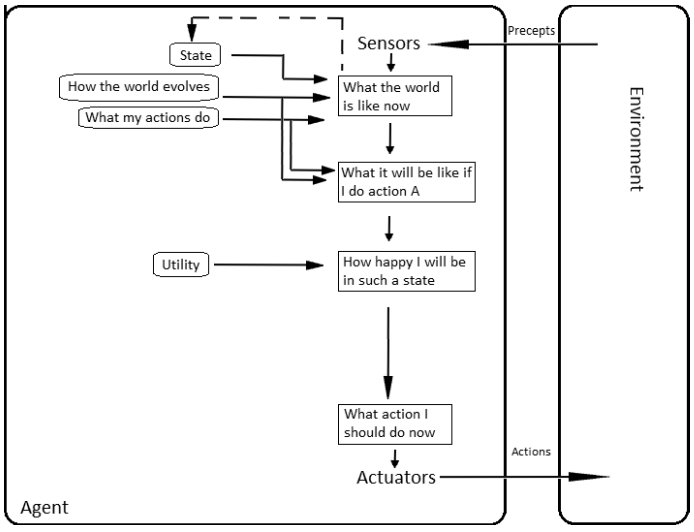
State
How the world evolves
What my actions do
Utility
Sensors
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Actuators
Agent
Precepts
Actions
Environment

Fig. 4: Utility-Based Agent Architecture [1]

possible actions and rank them in two ways, symbolically or numerically, based on their efficiency of reaching the goal, and the most efficient is executed. Symbolic comparisons are represented by action A is better than action B which is better than action C, or A ¿ B ¿ C, so action A will be performed. Some symbolic comparisons run into some issues when trying to determine which actions are better than others due to a large amount of dynamic variables in the environment. Some actions may be better according to some variables within the state then others, and vice versa. This adds a level of complexity that is not entirely solved yet, as far as symbolic comparisons go. Numeric comparisons are able to weight each action numerically based on its efficiency such as action A's numeric preference is 0.9, action B's numeric preference is 0.83, and action C's numeric preference is 0.74, so action A will be performed. This allows much more precision when calculating how much influence a certain action will have on the current state. For example, if a road is blocked off and the agent has to get to the other side, a goal-based agent will simply take the first route it realizes will succeed regardless of efficiency, as for the utility-based agent it will calculate the shortest route by processing all possible goal achieving routes and selecting the most efficient to actuate.

E. Learning Agents

Learning agents are able to explore unknown environments and create their own actions, given some initial knowledge. They are able to do so because of their four components: the learning element, critic, performance element, and problem generator [1].

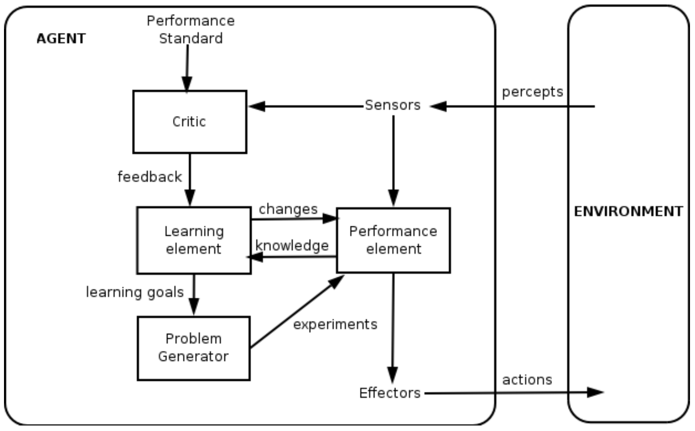

Fig. 5: Learning Agent Architecture [1]

The learning element is in charge of making changes to the agent's knowledge. It does so by analyzing past sequential states and making assumptions as to how to world evolves between states.
The critic is used to determine whether or not the agent is doing well and decides whether or not changes need to be made to the performance element. The critic often uses the weights from the utility agent to scale good and bad decisions based on their numeric differences.
The performance element is what decides the actions to perform from a set of actions it already knows. This component is what all of the previous agents resemble.
Lastly, the problem generator is what suggests new actions that will lead to new experiences based on the criteria of the agent's goals and its past states, referencing them to avoid duplication. All together these four components can simulate the basis of human learning.
An analogy often used to explain learning agents is a student taking a test. The student is testing its knowledge on a test, which is then graded by the teacher and returned to the student showing he/she what they could improve on. Apart from testing, one must also be exposed to new knowledge, so imagine a friend showing the student a new study technique or experiment that taught them something new to try. When the test comes back around, the student will have more knowledge and perform better. In this anology the student is the performance element, the test is the critic, the teacher is the learning element, and the friend is the problem generator.

III. Cognitive Architectures

Cognitive architectures are those that mimic the structure of a human mind through perception and learning. They are able to analyze the results of actions on the current world and construct a comprehensize computer model from it. This constructs what is known as ”working memory” or a representation of the current situation/state [4]. Cognitive architectures consist of different types of memories for storing knowledge, such as semantic, sequential/episodic, or procedural. They also contain processing units that extract, select, combine, and store knowledge. They each use certain languages for representing the knowledge that is stored and processed, generally in multi-dimensional vectors and matrices to hold all variables wihtin the current and past states [5]. Although hundreds of cognitive architectures have been created since the study of AI, we are going to focus more on John Laird's Soar architecture to gain a basic understanding of at least one cognitive architecture.

A. Soar

The Soar cognitive architecture was created by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University. It is now being researched further by John Laird's research team at the University of Michigan. Soar is not a model of working memory, but instead a cognitive architecture that focuses on the functional capabilities needed for a memory system to support performance in a range of cognitive tasks [4]. This key point is the reasoning for its multi-component architecture which will be discussed in detail. Learning within the working memory phenomena is the key feature of Soar because working memory cannot be studied independently of long-term memory while ignoring learning [4]. More details on learning techniques will be decribed in the Multi-Agent Systems section. Soar's main focus is how long-term memory acquired through prior task performance is involved with working memory.

1) Processing Cycle: Soar's processing cycle is mainly the cooperation between procedural memory (knowledge of how to do things) and working memory (knowledge observed from the current state) to help with the selection and application of operators. The working memory is represented within the system as a symbolic graph structure in the original soar but later upgraded to a numeric graph structure in Soar 9.

The procedural memory is composed of a set of condition-action rules and continuously matches the data of the working memory to its conditions, and if a condition is met, the action is performed. This is also known as a production system. Soar has a difference in their production system compared to others, because when Soar finds several conditions that are met by the working memory in the procedural memory, all of the actions are performed concurrently. This is known as the elaboration phase.

Next the system will simulate each applicable action by creating an operator in working memory as well as marking its acceptable preference to the current state. The acceptable preference are all symbolic in the first version of Soar, such as operator A is better than operator B, but in Soar 9 preferences are numeric, such as operator A is .84 and operator B is .82 [4], [5]. The conditions within the procedural memory will then be compared to each operator and will be marked as acceptable preferences if applicable. These acceptable preferences of the "future" conditions are then compared to see which original operator will be installed into the current working memory. This is known as the "Operator Selection and Operator Application" phase.

Actions that are associated with the chosen operator are executed and make changes to the working memory. These changes to the working memory can be any of the following: inferences, queries for retrieval from the long-term semantic or episodic memories, motor commands, or interactions with the working memory's interface to perception, Spatial Visual System (SVS). This is known as the "Output" phase.

These changes to working memory lead to new operators being proposed and evaluated, followed by the selection of one and its application. See Figure 6 for a visual representation of the original processing cycle or soar prior to adding episodic memory, semantic memory, and much more which can be seen in the next figure.

Figure 6 displays the original idea Laird had for his Soar cognitive architecture. Soar has recently been extended by adding new non-symbolic representations of knowledge along with associated processing and memory modules, and new learning and memory modules that capture knowledge that is cumbersome to learn and encode in rules [5]. The new version of Soar (Soar 9) is still working based on procedural

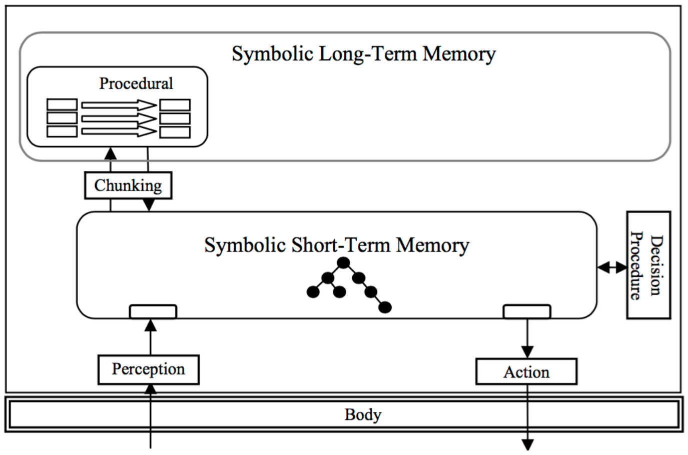

Fig. 6: Soar Processing Cycle [5]

knowledge, but the new components influence decision making indirectly by retrieving or creating structures in symbolic working memory that cause rules to match and fire [5]. The new components include working memory activation, which provides meta-information about the recency and usefulness of working memory elements; reinforcement learning; which tunes the numeric preferences of operator selection rules; the appraisal detector, which generates emotions, feelings, and an internal reward signal for reinforcement learning; semantic memory, which contains symbolic structures representing facts; episodic memory; which contains temporally ordered snapshots of working memory; a set of processes and memories to support visual imagery, which includes depictive representations in which spatial information is inherent to the representation; and clustering, which dynamically creates new concepts and symbols[5]. I will now go into detail about the components of the Soar architecture.

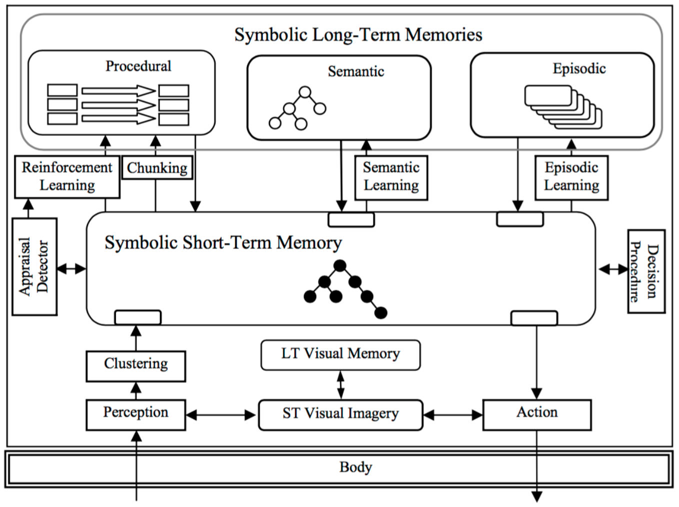

Fig. 7: Soar 9 Processing Cycle [5]

2) Working Memory Activation: The activation on the working memory allows the system to remember how recent and relevent a certain action is. These actions are chosen when Soar concurrently fires them. This data is stored in the episodic memory and is used for retrieval purposes only to assure that certain actions are still relevent to the system based on their environment. In the future episodic memory is seen to develop emotions within intelligent agents [6].

3) Reinforcement Learning: Reinforcement learning is the selection of actions to maximize reward. Prior to the new expansion of Soar 9, Soar used symbolic preferences to weigh out its options, such as action A is better than action B, but in Soar 9 Laird introduced numeric preferences which allow the operators to be compared in a more complex but more accurate manner. During operator selection, all numeric preferences for an operator are combined, and an epsilon-greedy algorithm is used to select the next operator [5]. After an operator applies, all of the rules that created numeric preferences for that operator are updated based on any new reward recieved from success or failure, and the expected future reward, which is simply the summed numeric value of the numeric preferences for the next selected operator [5]. The ability for multiple rules estimating the future reward of an operator allows hierarchical representation of rules and can speed up the learning process. Reinforcement learning simply modifies the numeric preferences of existing rules over time, while chunking can create new rules when encountering an impasse.

4) Impasses, Substates, and Chunking: What does a system with the Soar architecture do when it cannot come to a conclusion on the selection of a single operator? The system creates a substate in the working memory that has the desired goal in mind. Simulating procedural knowledge upon this substate will propose operators to be called within the substate. Keep in mind that the substate is not being actuated by the system, it is only internal. The chosen operators will create numeric preferences in the original state or lead the system to modify the current state so that the impasse can be solved. Substates provide a means for on-demand complex reasoning, including hierarchical task decomposition, planning, and access to the declarative long-term memories [5]. After the substate resolves the impasse, every component of the substate except for the result is terminated. The chunker then compiles the result into a set of rules that will lead to the result, and then performs them. The results of this impasse will be stored within the long-term memory so that no impasse will occur again for a similar situation.

5) Semantic Memory (SMEM): Also known as the long-term memory of a Soar agent, its main purpose is to record fact-like structures, such as tables have legs, dogs are animals, etc [5]. All data within SMEM is represented as directed cyclic graphs. This allows the agent to call upon its knowledge of an occurance to help make a decision in its current state. Like stated in the Working Memory Activation section, the SMEM record the frequency at which a structure is used. This technology was originally developed by the ACT-R cognitive architecture and is used to speed up the process of learning by keeping common operators nearby. The soar architecture also supports an activation method called spreading activation which allows recently activated structures that are in the working memory to activate the structures they are linked to within the SMEM. The SMEM has 3 core functions: Encoding, Retrieval, and Storage.

Encoding has to do with what the agent should choose to store within the semantic memory, when to add new declarative chunks, and how to update prior semanic rules. Retrieval deals with how the cue is placed and matched. When trying to figure out what to save the agent generally saves WMEs with the same identifier as a declarative chunk, or individual WMEs that serve as sementic links between chunks. When deciding when to add a new declarative chunk the semantic memory generally saves all WMEs that ever exist, or save them selectively through domain-independent criteria (activation) or deliberately via Save Link. When updating semantic memory it is possible to either overwrite old values with new values in a pre-existing chunk or create a new separate chunk with the updated values with the methods above.

Retreiving data from semantic memory is done by cue links. A cue link is a function that searches all semantic rules for a match to the current state. A cue link can be coded as the following:

```
(state <s>)(<s>..<cue>)
-->
<s>.memory.cue-link.cue <cue>
```

There are two different types of retrievals: global search and local information access. A global search is done by a Cue Link while a local information access is done by an expand link.

Storing data within semantic memory is conceptually the same structure as WMEs. The activations are maintained for individual attributes (WMEs), and long-term identifiers serve as the chunk name.

6) Episodic Memory (EPMEM): The Episodic Memory (EPMEM) is the memory associated with the experiences that the agent has had over time [7]. In Soar the EPMEM links instances of memory structures that occured in the working memory at the same time, allowing them to remember the context of past experiences as well as the relationships between experiences [5]. The system creates a buffer of the current working memory, finds the best partial match it can within the EPMEM (biased by frequency and working memory activation), and inserts the match into a separate working memory buffer to distinguish it from the current situation [5]. Once loaded into its separate buffer, each proceeding episode to the retrieved memory will be added to the buffer sequentially. The actions are then carried out and the agent will successfully perform the same actions again. Episodic memory works hand-in-hand with impasses and are very powerful when it comes to consistency and accuracy of agents in common states.

7) Spatial Visual System and Mental Imagery: The Spatial Visual System (SVS) is the eyes of the Soar agent. The SVS creates a scene graph that contains everything it currently sees, along with their spatial properties such as shape, location, pose, relative position, and scale [8]. The SVS can also insert hypothetical objects into the current scene to make mental calculations based on sight and procedural knowledge, this method is used with movement simulation and planning. The SVS reports the state of the environment to the working memory by a Scene Graph Edit Language (SGEL). The SVS organizes objects as a tree of nodes showing their relationship and how they're connected. Each node within the tree has a position, rotation, and transform (reletive to its parent node). Here is what a sample of SGEL code would look like decribing a scene with a car and a pole.

```
(S1 ^svs S3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world ^child C1 ^child C2)
(C1 ^id pole)
(C2 ^id car ^child C3 ^child C4
 ^child C5 ^child C6 ^child C7)
(C3 ^id wheel0)
(C4 ^id wheel1)
(C5 ^id wheel2)
(C6 ^id wheel3)
(C7 ^id chassis)
```

This graph of the current environment will then be used by the working memory to influence its decision making by providing the spatial relations between objects. Caches within the SVS temporarily hold scene graphs to avoid redundant computations. When a substate is created a copy of the current visual state tree is copied to each substate and altered independently from the original within that substate. Having a visual understanding of the world increases the accuracy of an agents decision making by basing its knowledge off of more than just basic sensor data. Figure 8 shows the proccessing within a SVS.

8) Overview of Soar: Having an understanding of the basics of a cognitive architecture will help when thinking of the big picture. The components listed above are not the only way to accomplish autonomous learning, many other studies have accomplished a similar goal with varying components. Now that we know the power of a single intelligent agent, we will discuss how using multiple cognitive machines can benefit our research into AI.

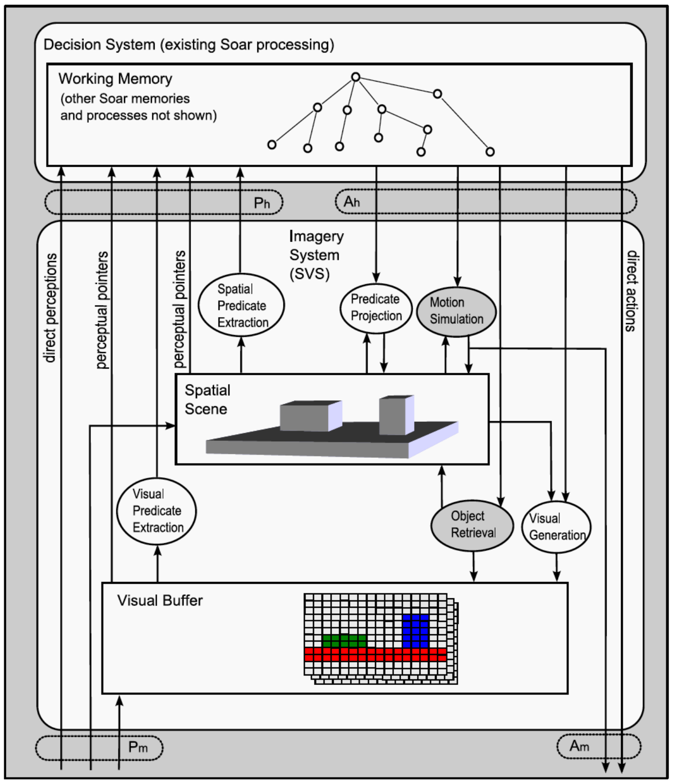

Fig. 8: SVS System design. Boxes are short-term memories, circles are processes. Grey circles involve access to information in long-term memory (knowledge). There are implicit control lines (not shown as arrows) between working memory and all of the processes shown [8].

IV. Multi-Agent Systems

The definition of a Multi-Agent System (MAS) is yet to be set in stone. Some researchers say it is the use of multiple intelligent agents to solve a problem that is far too large or complex for one agent to solve, others say that it is the use of one intelligent agent and several followers mimicking its advice about the environment. The fact is that MASs are relitively new to the computer science community and nobody is aware of it's functionality or potential, yet.

According to Wooldridge, MASs must have three characteristics, ther must be autonomous, no agent has a full global view of the system, and there is no designated control agent [9]. I believe that the following are areas of MASs that need to be researched in order to obtain new knowledge: environmental characteristics, heterogeneity vs. homogeneity, cooperation, inter-agent communication, and learning techniques. Other possible thoughts on MASs will be discussed in a later section.

A. *Environmental Characteristics*

One of the key characteristics of MASs is that no agents has complete knowledge of the environment. In other words, each agent can only have knowledge about its own current state and is unaware of other's [10]. This poses many problems for agent cooperation. The following are characteristics of the environment: accessability, determinism, dynamics, discreteness, dimentionality, and whether it is episodic [11]. I will be using an example of a pitcher in a baseball game to describe each characteristic and how each effects the complexity of the experiment.

Accessibility determines how easily the environment is to be read [11]. In a game of baseball there are a finite number of variables that are recorded within each state such as number of outs, number of strikes, number of balls, number of opponents on base and their location, therefore this environment is very accessible for the pitchers decision making and does not contain too much noisy data.

Determinism is whether or not there is a specific response to some actions [11]. In our example there are very few instances where there is only one choice of action, such as walking to the dugout when three outs are reached. The constant decision between actions is what makes the game so variable. Therefore, the determinism of a baseball game is low but higher than that of the open world.

The dynamics of the environment is how many entities are influencing the environment regularly [11]. In our example the only thing that can happen to the environment before a pitcher throws the ball is batter stepping out of the batters box and calling time out, or a runner stealing a base, other than those two options, the current state will not change unless the pitcher throws the ball.

Discreteness is if the environment allows a finite number of possible actions [11]. This differs from determanism because it isn't a specific response to an action it is the size of the set of actions you can choose from. In our example, the pitcher can either throw the ball or step off of the mound when starting a pitch, that means in this current state there are only two choices composing a finite set of actions.

The dimensionality of the environment is whether the amount of space if valued in the decision making of the agent. In baseball, the distance of the ball from a defender determines if they are able to catch it or if they need to realign themselves for the catch. The combination of all of these characteristics determine how complex of an environment the agents are working within and how this impacts the accessability of resourses and multi-agent coordination [11].

If an environment is episodic, which most are, then actions on the current state affect the future [11]. Everything about a baseball game is impacting the future actions, from the types of pitches, to the number of hits by the batters, but the problem with this type of environment is that it is not certain that if one action is performed that a specific episodic action will occur after.

B. *Heterogeneity vs. Homogeneity*

Heterogeneity and homogeneity are terms that describe the composition of agents and their knowledge within a multi-agent system. To help understand the terms better I am going to start off with the example of having two children in a room trying to accomplish a task. One child has the math problem 2+2 in front of him and the other has a short story in front of him. Imagine a world where it is possible for global knowledge to exist, no matter which child accomplishes which task, both have the knowledge from it. The children now have knowledge of both 2+2 and the short story resulting from individual efforts, this classifies them as homogeneous agents because of their identical knowledge set. Now think of the real-world where everyone has their own intelligence level and understanding of events based on experience, in this case we are classified as heterogeneous agents with varying knowledge sets.

The way to assemble homogeneous agents is to make one learning agent that communicates with a finite set of non-learning agents that have fixed knowledge of the environment. Multiple learning agents can exist in multi-agent systems, but this introduces game-theoretic issues to the learning process which are not yet fully understood [12]. The ability to sanction off "teams" of agents with identical knowledge sets allows for better cooperation.

C. *Cooperation*

Most multi-agent systems base their cooperation requests off of direct communication through weighted request matrix and a weighted response matrix. When one agent sends a weighted request matrix to all heterogeneous agents (all homogeneous agents can accomplish the same task as itself) then a weighted response matrix is sent back from all qualified agents with a numeric cost, where the lowest cost will be chosen followed by the formation of a contract for the current task.

Another method of cooperation requesting is called the "pheromone" method, which is performed through indirect communication. In this method, agents leave a request at its location and the next agent in the vicinity will pick up the request and help if possible or leave it for the next agent to access. These types of requests can dissolve over time or grow in urgency based on the context of the cooperation request [12].

D. Inter-Agent Communication

Communication is the altering of the state of the environment such that other agents can perceive the modification and decode information from it [12]. Agents can send various data types through multiple methods. As mentioned in the previous section there are direct and indirect communication methods. Both are used in multi-agent systems and aid cooperation. Communication between agents must have three fundamental components: a common language, a common understanding of the language exchanged, the ability to exchange [13]. Below is a figure showing the components used for agent communication in a multi-agent system.

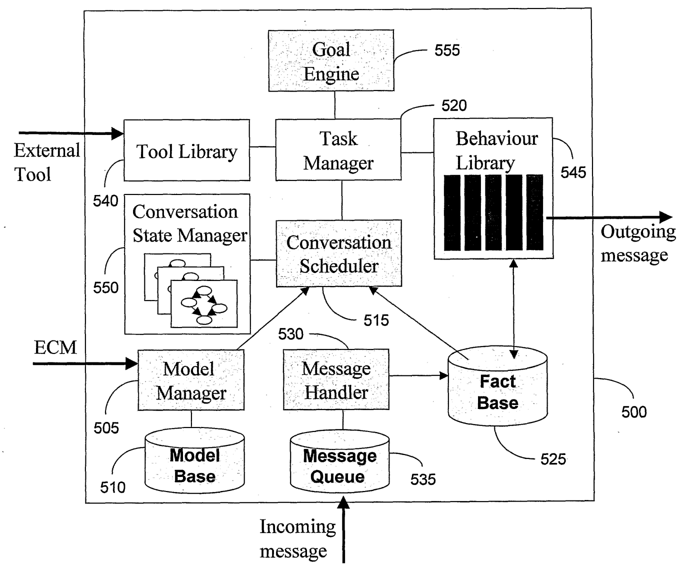

Fig. 9: Agent communication components [12]

The requirements for communication languages are: form, content, semantics, implementation, networking, environment, and reliability.

The form of the language should be easily readible by agents and by people. It should be declarative and syntactically simple. The readability should not impact its conciseness, ability to parse, or its ease to generate. The language has has to be linear or easily translated to a linear form in order to send through a transport mechanism, and the syntax should be extensible [13].

The content of the language should be easily distinguishable, from a finite set of communicative acts (primitives) to content messages about the domain. Having a finite set of primitives will allow portability to different types of systems. Having a finite set of primitives also allows the commitment to a specific content language, which allows for more complexity on the content language level. The disadvantage to this is that every application must use the same content language so messages do not have to be translated. This is a very large constraint [13].

The semantics of the language is often limited to naturaal language descriptions, and a more formal description is needed if the language is intended to communicate over a diverse range of applications. Semantic properties of a communication language should follow those of normal programming languages, meaning it needs to be grounded in theory and unambiguous. Due to the environment of a communication language, the language must be able to take time and location and address them into the semantics. Lastly, the semantic description should provide a model of communication, which would be useful for performance modeling [13].

The implementation should be efficient in speed and memory utilization. The details of networking layers should be hidden to the user. The imformation within each message should be amenable to handle certain subsets of comminicative acts [13].

The language should follow modern networking technology because some communication will be about concepts involving networked communications. It should support point-to-point, multicast, and broadcast. Synchronous and asynchronous connections should be supported. High-level protocols should be independent of the transport mechanism used, such as TCP/IP, email, http, etc [13].

The environment will be highly distributed, heterogeneous, and extremely dynamic. Therefore, the language must have conventions to cope with heterogeneity and dynamism. It must support interoperability with other languages and protocols. It must support knowledge discovery in large networks. It must be easily usable in legacy systems [13].

The language must be reliable. Communication must be secure. Private exchanges between two agents should be supported. A method of authentication of agents should also be present. Any malformed or inappropriate messages should be dealt with, and error handling should be reasonable [13].

E. The Basics of Agent Communication Languages (ACL)

The languages used for inter-agent communications are often known as Agent Communication Languages or ACL. As stated before, ACL's main goal is to exchange information and knowlegde. Many methods of agent communication have been tested and failed, such as remote procedure call (RPC), remote method invocation (RMI), CORBA, and object request brokers. ACLs are different from the previous attempts at seamless communication for two reasons: ACLs handle propositions, rules, and actions instead of simple objects with no semantics associated with then, and ACL messages describe a desired state in a declarative language, rather than a procedure or method [14]. Now just because ACLs solved some issues from previous tests, this does not mean that research of an effective inter-agent language is finished. ACL is far from optimal and most of the problems with multi agent systems come from their communication. Agents should be able to not only share propositions, rules, and actions, they should be able to share plans, goals, shared experiences, or even long term strategies.

ACLs ransport messages over the network using a lower-level protocal such as SMTP, TCP/IP, IIOP, or HTTP. Agents generally do not communicate in single messages. Like humans, agents must communicate in conversations in order to get all information about the scenario across to the other agent. These topics often range from negotiation to auctions for cooperation contracts. There are two well known languages that have been repurposed as an ACL, they are the Knowledge Query and Manipulation Language (KQML) and the FIPA ACL.

KQML is not only a messaging format, but also a message-handling protocol that supports run-time knowledge bewteen agents. KQML is genally made up of three layers: the content layer, the communication layer, and the message layer [15].

The content layer holds the actual content of the message wanting to be sent, represented in the porgrams own representation language. KQML can carry expressions encoded in any representation language, for example ASCII strings [13]. Some KQML-speaking agents ignore the content portion of the message.

The communication level encodes a set of message features which describe the lower level communication parameters, this includes the sender identity, the recipient identity, and also a unique identifier associated with the communication. The message layer is used to encode a message wanting to be sent out to another application, this is also known as the ”core” of the KQML language.

The message layers primary function is to identify the protocol that is going to be used to send the message and to supply a speech act or performative which the sender attaches to the content [14]. The syntax of KQML is based on a balanced parenthesis list. The first elemment in the list is the performative; the remaining elements are the performative's arguments as valued pairs. Below is an example of a message written in KQML from agent joe asking another agent for a query about the price of a share of IBM stock:

```
(ask one
 :sender joe
 :content  (PRICE IBM  ?price)
 :receiver stock-server
 :reply-with ibm-stock
 :language LPROLOG
 :ontology NYSE-TICKS)
```

In this example the :content (PRICE IBM ?price) is the content layer, the :reply-with ibm-stock, :sender joe, and :reciever stock-server are the communication layer, and the :language LPROLOG and :onthology NYSE-TICKS are a part of the message layer [13]. After the recipient recieves the message joe might recieve a message looking like this:

```
(tell
:sender stock server
:content  PRICE IBM
:receiver joe
:in-reply-to ibm stock
:language LPROLOG
:ontology NYSE-TICKS)
```

An agent can also send a request to all available agents with the following ask-all protocol [15]:

```
(ask-all
:content  "price(ibm,(Price,Time))"
:receiver stock-server
:language standard_prolog
:ontology NYSE-TICKS)
```

The stream-all performative asks that a set of answers be turned into a set of replies [15]. The following is how it would be written in KQML:

```
(stream-all
:comment "?VL is a large set of symbols"
:content (PRICE ?VL ?price))
```

An agent can also put the recieving agent on hold while waiting for the next message from the sender by nesting a command within the standby performative [15].

```
(standby
:content  (stream-all
:content (PRICE ?VL ?price)))
```

To receive all different set of answers including the updated versions of the answers that will present themselves in the future you have to use the subscribe performative [15].

```
(subscribe
 :content (stream-all
 :content (PRICE IBM ?price)))
```

FIPA ACL is a very commonly used language and is currently used in the Java Agent DEvelopment Framework, also known as JADE. FIPA is based on two assumptions. The first assumption is that the time that it takes for agents to reach a consensus should not be long. The second is that only external behaviors of the system components should be specified [16]. This allows all of the internal interface of the agents to be left to the agent developers. Figure 10 shows a sample communication diagram using the reply, query, and subscribe performative in KQML.

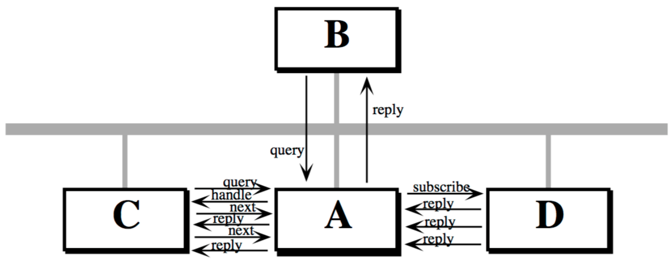

Fig. 10: Sample Communication Diagram using query, reply, and subscribe in KQML [15]

F. Learning Techniques

There are three types of learning techniques: supervised, unsupervised, and reinforced. We touched briefly on reinforced learning when discussing the Soar cognitive architecture, and now we will be comparing it with other forms of learning.

The least common form of learning is supervised learning, this is because the agent requires the critic to always have the correct answer for the situation at hand, and after it is told once, it knows what to do if the state arises again. This method is generally used during the initial training phase to teach the agent some semantic or episodic rules of the environment. Supervised learning is a machine learning technique with applications in Handwriting Recognition, Speech Recognition, Pattern Recognition, and much more.

In unsupervised learning the agent performs actions without any feedback or evaluation of accuracy. This is not ideal for agent learning because we need to know which actions are more beneficial than others in certain states, and this method of learning only shows which actions can by executed in certain states without any learning.

The last and most popular is reinforcement learning. Reinforcement learning is another machine learning technique that is inspiredby behaviorist psychology. As stated earlier the critic in these situations provides either a symbolic or numeric feedback to weight each action based on how close it gets the agent to their goal. The figure below shows the flow of data through reinforcement learning.

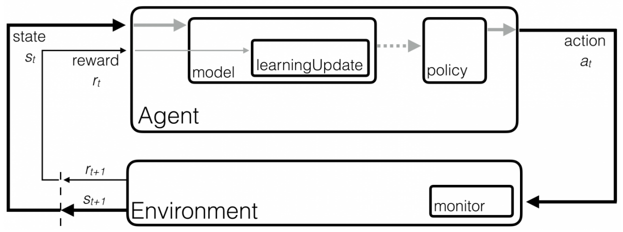

Fig. 11: Reinforcement Learning Cycle [17]

A basic reinforcement learning model consists of the following:

1) a set of environment and agent states S
2) a set of actions A of the agent
3) policies of transitioning from states to actions
4) rules that determine the scalar immediate reward of a transition
5) rules that describe what the agent observes

With reinforcement learning in multi-agent systems you can either have one learning agent and several followers (per group), this method is called team learning, or you can have concurrent learning agents in the same group [12]. The more learners that there are in a group, the more noisy the data being communicated will be. Having too much heterogeneity causes many problems arise which we will discuss in the next section on calibration and conscensus problems.

V. Consensus Problems

When observing the natural world from an given state there will always be sets of incomplete information that an agent cannot read. One of these things being the current state and the decisions of others. The problems this brings up in multi-agent systems is called consensus, or the inability to agree on the next action with other agents. This difficulty can be traced back to a well known theory called Game Theory researched by the father of computers, John Von Nuemann.

Before we discuss game theory we will introduce decison theory. Decision theory is means of analyzing which of a series of options should be taken when it is uncertain exactly what the result of taking the option would be [18]. Descision theory's main focus to to choose the "best" action to perform, where best generally is defined to maximize the expected utility of the decision maker. Decision theory provides a strong foundation of the decision process of an agent within an unpredictable environment.

Game theory [19] branches from it's sister theory decision theory, the difference is that it accounts for how multiple self interested agents play a role in making a single decision, opposed to what is the "best" option for one agent. It can often be described as the mix of probability, consequences and nature. Game theory still tries to find the best action to perform, but takes into account the best option for other agents as well. Game Theory focuses on the problems of how interaction strategies can be designed that will maximise the outcome of an agent in a multi-agent encounter, and how protocols or mechanisms can be designed that have these desirable properties [18]. Decision theory can be described as a game against nature, where nature is an opponent that is not seeking to gain the best outcome, but rather acts randomly. When the opponent and the "player" of a game has to negotiate or coordinate with its opponent to assure that both recieve an optimal outcome, this is game theory in multi-agent systems.

One of my favorite examples to model the logic of game theory is called the prisoner's dilemma. In this scenario there are two prisoners that have two options, to confess or to remain silent. If both prisoners confess they both recieve five years in prison. If one confesses and the other remains silent, the one that remains silent recieves twenty years and the one who confessed recieves no time at all. And finally if both remain silent they both recieve only one year. So in this example there are only two choices to choose from, but in a mult-agent environment there will me lots of choices, increasing the complexity of each decision. The hard part to determine about this scenario is which outcome is considered optimal. Does the system want to benefit only one of the agents, or does it want to choose an equal outcome for both. This is where human nature takes over for this game theory problem in the real world, but in a multi-agent environment it is extremely difficult to come to an optimal decision. Below is a visual representation of the prisoner's dilema.

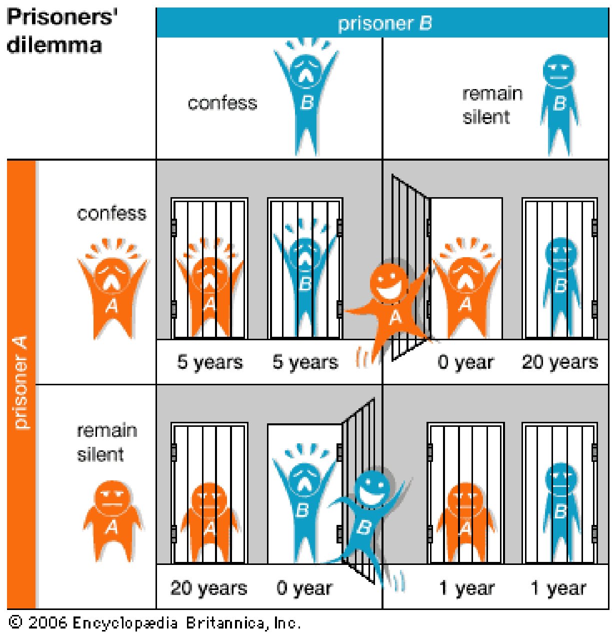

Fig. 12: Prisoner's Dilema image from the Encyclopedia Britannica

I will not be going into the details of game theory due to its complexity, but for more information about the different types of game theory I recomment reading the book *A Course in Game Theory* by Martin J. Osborne [20].

VI. Is Omniscience Possible?

Omniscience is the state of knowing everything, so could it be possible for agents to someday be able to know everything, including the current states of other agents? Omniscient agents are those who know the outcome of their actions, everytime, before they execute. Currently all agents are rational which means they make decisions based on rationalizing their possibilities and picking the best. Technology has advanced to levels we never imagined possible, so could humans create the first omniscient system? Here are some possibilities I can see becoming possible over the next generation of research.

Is the internet creating a database for an omniscient agent? It is a known fact that you can find relatively anything on the internet, and recently companies such as Amazon, with the Amazon Echo, and Google, with the Google Home, are creating AIs that reference the internet to fulfill requests with maximum accuracy. These devices don't have to observe the environment which their being questioned about, such as the weather in China, being in the United States there is no possible way for the Amazon Echo or the Google Home to know from observing its surrounding environment. Instead it referenced the internet which I refer to as a "pseudo-environment" and recalls the answer to you in an instant. With the internet being only a couple decades old, there is still plenty of room for growth. I strongly believe that sooner or later an intelligent agent will be able to know practically everything about the world based strictly on its internet access similarly to how humans have quicker access to knowledge than they did half a century ago. So just ask yourself, is omniscience possible?

Now we know that heterogenous team building is one of the key concepts of multi-agent systems but what if it is possible to only link the long term memories of all agents. This would make it so all agents within a multi-agent system would share their semantic, episodic, and procedural memory. With this capability when any agent learns something new it will be updated in the global library shared by all agents. Technically this would not be onmiscience because all agents are still unaware of the current state of each other agent and also the parts of the environment that are unmeasurable during its current state. I believe that implementing a shared long term memory would improve on the learning curve for every agent. This organization of memory allows each agent to know the same facts about the environment and allows cooperation requests to just call on the nearest agent instead of sending out contracts and seeing which agent is smart enough to offer help. This constructs an unspoken trust between agents. Division of tasks can just be divided by the number of agents without deciding a heirarchy of tasks and which agent will perform each of them. The reason for only combining long term memory is because this memory is not contantly updated like the working memory or current state of the agent. This will allow the storage of episodes and semantic facts of the environment to be safely stored and accessed. The agents will now be homogeneous as far as knowledge, but will operate in perfect cooperation and have an increased learning rate. The scalability of this is unknown due to the unknown effects of multiple agents fetching data from a global memory at the same time, but I believe that at a low agent count, this method would succeed. The modern advancement that I believe will support this theory is cloud computing.

Maybe it is not possible to know everything about the world but is it possible for an agent to know everything about a specific human. This idea is influenced by an episode from the Netflix series "Black Mirror" about a technologically advanced dystopia in the near future. Think about creating an agent that has all of the same knowledge as you and allowing it to learn and make decisions based on your own actions. This agent would perform functions for you that it knows you would prefer before you tell it to do so. In Black Mirror this technology is used to control functions throughout the home such as cooking a breakfast you like at a certain time in the morning, playing a song throughout your home that you have listened to a lot lately, adjust the thermostat to your perfered temperature, etc. It's almost as if this agent is looking over you omnisciently throughout your day making sure that nothing that you dislike is going to happen because the agent dislikes those outcomes as well. So another theory is to what degree we want omniscience, on a full scale world level, a multi-agent cooperative team level, or a personal individual level.

VII. Conclusion

Multi-Agent Systems are far from perfect at this point in time, but their potential of Distributed Artificial Intelligence is incredible. The current issues with multi-agent systems lies within the communication system, from the language used, consensus, and communication delays. We have reached a point where we can make a self learning agent, but have yet to establish a consistently accurate universal agent language and communication method to make multi-agent systems completely viable and accurate. Once communications are constructed, cooperation requests can be modified and consensus issues can be further worked on. Multi-agent systems are extremely powerful and will emerge within society as soon as they are optimal.

With technology is getting smarter by the day, and soon all forms of technology will be able to communicate and cooperate together intellectually, this brings me back to my question, is agent omniscience possible? Will we ever reach a point where technology will know everything and humans will no longer need to learn for themselves? I believe this concept is going to become prevalent in the near future. These systems will likely take over many jobs, eliminating the possibility for human error. Society will have to think of new solutions for providing jobs or other financial benefits such as universal pay, mentioned by Elon Musk in early 2017, in a computer driven world. This is a sacrifice we have to be willing to accept in order to keep the advancement of technology going and discover it's true potential. All of this starts with further research of monolithic artificial Intelligence machines and expanding them into multi-agent systems.

References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed., 2003.

[2] M. Wooldridgey and P. Ciancarini, *Agent-Oriented Software Engineering: The State of the Art*, pp. 1–28. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[3] N. Vlassis, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Morgan and Claypool Publishers, 1st ed., 2007.

[4] A. Miyake and P. Shah, *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. Cambridge University Press, 1999.

[5] J. E. Laird, "Extending the soar cognitive architecture," *Frontiers in Artificial Intelligence and Applications*, vol. 171, p. 224, 2008.

[6] A. M. Nuxoll, *Enhancing intelligent agents with episodic memory*. PhD thesis, Citeseer, 2007.

[7] M. A. Wheeler, D. T. Stuss, and E. Tulving, "Toward a theory of episodic memory: the frontal lobes and autonoetic consciousness.," *Psychological bulletin*, vol. 121, no. 3, p. 331, 1997.

[8] S. Wintermute, "An overview of spatial processing in soar/svs," in *An Overview of Spatial Processing in Soar/SVS*, vol. CCA-TR-2009-01, University Of Michigan, 2009.

[9] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[10] D. Keil and D. Goldin, *Indirect Interaction in Environments for Multi-agent Systems*, pp. 68–87. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[11] D. Weyns, A. Omicini, and J. Odell, "Environment as a first class abstraction in multiagent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 14, no. 1, pp. 5–30, 2007.

[12] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, Nov. 2005.

[13] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "Kqml as an agent communication language," in *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, (New York, NY, USA), pp. 456–463, ACM, 1994.

[14] Y. Labrou, T. Finin, and Y. Peng, "Agent communication languages: The current landscape," *IEEE Intelligent Systems*, vol. 14, pp. 45–52, Mar. 1999.

[15] T. Finin, R. Fritzson, D. P. McKay, and R. McEntire, "KQML - A Language and Protocol for Knowledge and Information Exchange," in *13th Int. Distributed Artificial Intelligence Workshop*, pp. 93–103, AAAI Press, July 1994.

[16] F. Bellifemine, A. Poggi, and G. Rimassa, *JADE - A FIPA-compliant agent framework*, pp. 97–108. The Practical Application Company Ltd., 1999.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[18] S. Parsons and M. Wooldridge, "Game theory and decision theory in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 5, pp. 243–254, 2002.

[19] K. Binmore, *Playing for Real: A Text on Game Theory*. Oxford University Press, 2007.

[20] M. J. Osborne and A. Rubinstein, *A course in game theory*. Cambridge, USA: The MIT Press, 1994. electronic edition.