A Survey on Software Agent Architectures

Rosario Girardi and Adriana Leite

Abstract—**An important decision when developing a software agent is the design of its internal architecture. Several models of deliberative and reactive architectures have already been proposed. However, approaches of hybrid software architectures that combine deliberative and reactive components, with the advantages of both behaviors, are still an open research topic. This paper analyzes the state of the art of software agent architectures from the basic reactive and deliberative models to more advanced ones like hybrid and learning software architectures. A case study on the design of an ontology-driven hybrid and learning software agent architecture is also described.**

Index Terms—**Software Agents, Software Architectures, Hybrid Agents; Learning Agents; Agent-oriented Development; Software Design**

I. Introduction

DEVELOPING software of high quality is difficult because of the natural complexity of software. Looking for appropriate techniques to confront complexity, software development paradigms have evolved from structured to object-oriented approaches to agent-oriented ones [31].

Having the properties of autonomy, sociability and learning ability, software agents are a very useful software abstraction to the understanding, engineering and use of both complex software problems and solutions like distributed and open systems and to support the decision making process [19][23]. A software agent is an entity that perceives its environment through sensors and acts upon that environment through actuators [38]. Agent attributes allow to approach complexity of software development through appropriate mechanisms for software decomposition, abstraction and flexible interactions between components [31].

During the process of developing a multi-agent system, both the architecture of the agent society and the one of each agent are defined in the global and detailed design phases, respectively, looking for satisfying the functional and non-functional requirements of the system.

A software architecture is a software computational solution to a problem showing how the component parts of a system interact, thus providing an overview of the system structure. It is the product of a software design technique and considered a bridge between requirements engineering and coding where emphasis is on coordination and cooperation over computation. The main elements of a software architecture are components, connectors and cooperation and coordination mechanisms. The components are the computational units, like modules, objects or software agents. The connectors represent the interactions between the architecture components, for instance, the messages exchanged by agents in a multi-agent society. The cooperation and coordination mechanisms define the way in which the elements are arranged, for example, a layered architecture.

Rosario Girardi and Adriana Leite are with the Computer Science Department – DEINF/GESEC, Federal University of Maranhão, Brazil (E-mails: rosariogirardi@gmail.com, adri07lc@gmail.com).

This work is supported by CAPES and FAPEMA, institutions of the Brazilian Government for scientific and technologic development.

Software architectures are represented in graphical diagrams based on architectural styles and design patterns. An architectural style [20] defines a vocabulary of components and connection types and a set of restrictions on how these components may be combined. A design pattern [35] is a reusable solution to a recurring problem. It shows not only the solution but also its restrictions and the context in which to apply this solution. Frequently used architectural styles and design patterns are pipes and filters, object-oriented, layered and blackboard architectures.

Agent architectures emulate human behavior through models of reactive architectures, supporting instinctive or reflexive behavior, and deliberative architectures, supporting different forms of automatic reasoning (deductive, inductive and analogical reasoning, among others).

A reactive architecture is ideal in cases where an immediate action is necessary for a certain perception, and then, its main advantage is the speed of the agent action. Differently, a deliberative architecture is suitable to support more complex decisions where a reasoning process should be executed to find the most appropriate action for a particular perception.

Several models of deliberative and reactive architectures have already been proposed. However, approaches of hybrid software architectures that combine deliberative and reactive components, with the advantages of both behaviors are still an open research topic. These architectures have greater complexity in their definition and use, since they require synchronization between reactive and deliberative components. Another important aspect to be considered on the definition of the internal architecture of an agent is the definition of knowledge bases representing the agent knowledge about itself and the external environment. One of the most effective ways of representing agent knowledge bases are ontologies. An ontology is an explicit specification of a conceptualization [39]. Conceptualization refers to the abstraction of a part of the world (a domain) where are represented the relevant concepts and their relationships. Ontologies have the advantages of being formal and reusable semantic representations.

Considering that an agent has the granularity of a subsystem, the specification of its architecture has particular importance both for approaching complexity and understanding software solutions.

This article analyzes the state of the art of software agent architectures from the basic reactive and deliberative models to more advanced ones like hybrid and learning software architectures.

The rest of the paper is organized as follows. Section II introduces the basic software agent architectures and section III their main components. Section IV discusses current approaches for advanced architectures such as learning and hybrid ones. Section V concludes the paper.

II. BASIC SOFTWARE AGENT ARCHITECTURES

Basic software agent architectures currently structure just two types of behaviors: reactive or reflexive and deliberative by reasoning. However, there are different proposals for structuring these basic behaviors and its variations [21][38] that are discussed in the following sections.

Fig. 1 illustrates the architecture of a generic agent, which perceives the external environment through sensors and then interprets the perception and transforms it into a sentence. It performs a mapping of the current perception to an action, represented in Fig. 1 by the question mark. This mapping occurs immediately in the case of a reactive behavior or through automatic reasoning otherwise. After encountering the sentence that represents the action, this is interpreted and the action is executed in the environment.

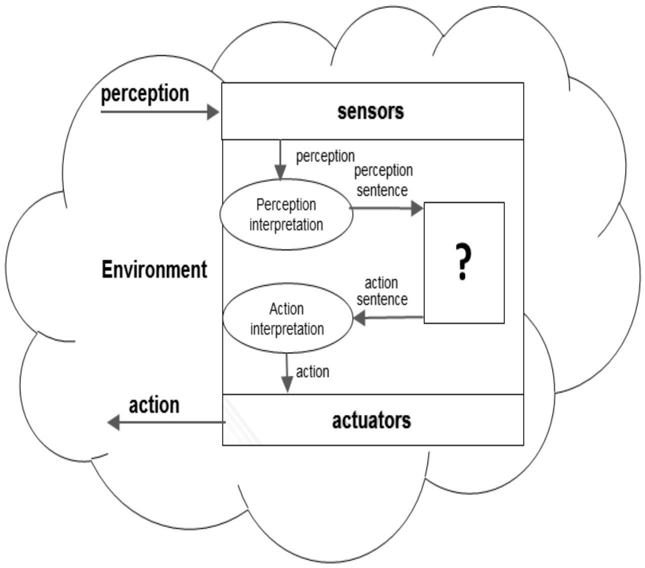

Fig. 1. A generic software agent (adapted from [38])

A. Russel and Norvig Classification

Russel and Norvig [38] define four basic types of architectures for software agents: simple reflex agents, reflex-agents with state, goal-based agents and utility-based agents.

1) Simple reflex agents

Simple reflex agents are considered the simplest type of agent where the agent's action is performed as an answer of just the current perception. It has a knowledge base that contains a set of <condition,action> rules. For each perception satisfying the condition, a corresponding action in the knowledge base is selected and performed. For instance, if the knowledge base has the reactive rule “if the car in front is breaking then initiate braking”, when perceived that “The car in front braked” the action “Initiate braking” would be performed. Fig. 2 and Fig. 3 illustrate the structure of a simple reflex agent. First, the perception is interpreted and transformed into a sentence, and then the knowledge base of the agent is updated. If a matching is found between the sentence and the condition of a rule, the corresponding action sentence of the rule is selected. Finally, the sentence that represents the action is interpreted and the agent performs the action in the environment.

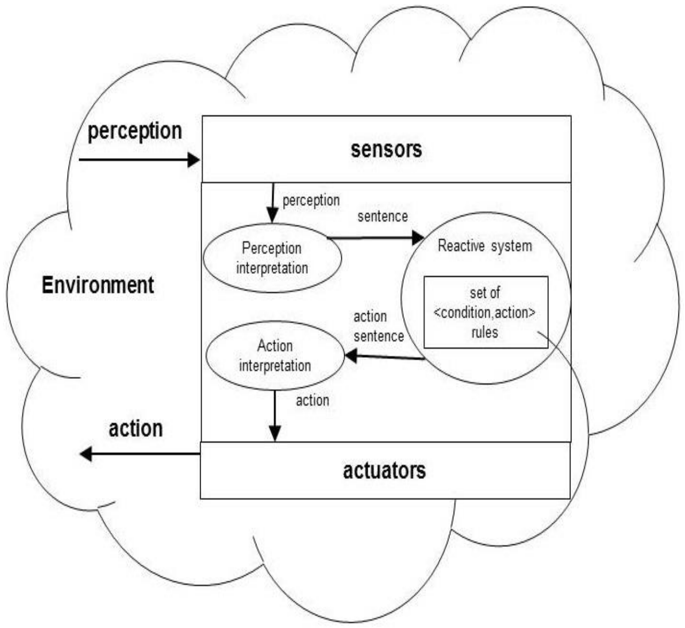

Fig. 2. The structure of a simple reflex agent (adapted from [38])

```
function Skeleton-Agent(percept) returns action
static: memory, the agent’s memory of the world
memory ← Update-memory(memory, percept)
action ← Choose-Best_action(memory)
memory ← Update-memory(memory, action)
return action
```

Fig. 3. Reflex-agent algorithm [38]

Fig. 4 illustrates a vacuum world as an example of a simple reflex agent architecture. This agent environment has two rooms called A and B. Each room can have a dirty or clean state. When the agent perceives that the room A is dirty it should perform the action "clean", then goes to the room B and do the same check. The operation of the vacuum agent is described in the algorithm of Fig. 5.

.

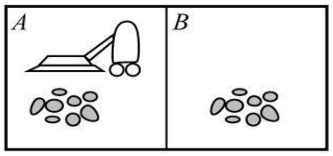

Fig. 4. The vacuum world: an example of a simple reflex agent [38]

```
function REFLEX-VACUUM-AGENT([location,status])
returns an action
 if status = Dirt then return Suck
 else if location = A then return Right
 else if location = B then return Left
```

Fig. 5. A simple reflex vaccum agent algorithm [38]

2) *Reflex agents with state*

A reflex agent with state differs from the previous one on the fact that it updates the state of the environment with each new perception. An action in the set of rules is selected according to the perception history and not just from the current perception (Fig. 6 and Fig. 7). Moreover, the agent with state has a world model, that is, contains knowledge about the environment. For example, when an agent crosses a street its reactive behavior will depend on the traffic rules that make part of the agent world model.

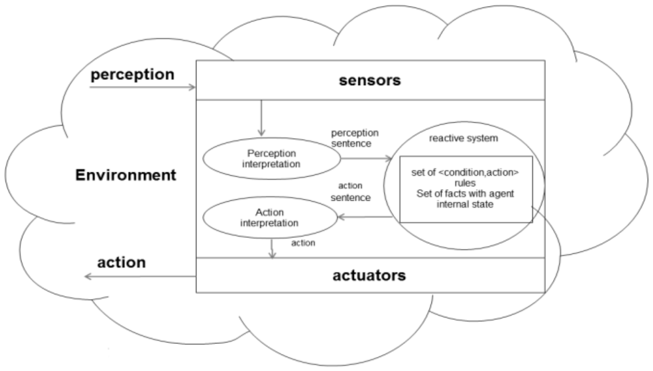

Fig. 6. The structure of a reflex agent with state (adapted from [38])

```
function Reflex-Agent-With-State(percept) returns action
static: state, a description of the current world state
        rules, a set of condition-action rules
 state ← Update-State(state, percept)
 rule ← Rule-Match(state, rules)
 action ← Rule-Action (rule)
 state ← Update-State(state, action)
 return action
```

Fig. 7. An algorithm with the basic operation of a reflex agent with state [38]

3) *Goal-based agents*

A goal-based agent maintains the state of the environment and has a goal to be achieved. For that it has to perform an action or a sequence of several actions determined though a reasoning mechanism on the agent knowledge base. This makes them less efficient than reflex agents due to the fact that the processing time required to perform a reasoning process is usually greater than the one required by rule condition-action agents. In Fig. 8 the basic structure of goal-based agent is illustrated and Fig. 9 describes the algorithm with the basic operation of this agent type.

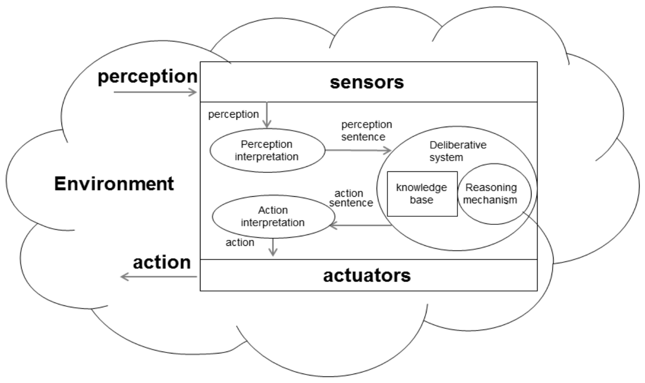

Fig. 8. The structure of a goal-based agent (adapted from [38])

```
function  Goal-Based-Agent (percept) returns action
static: KB, goal, search-space
 KB ← UPDATE-KB (KB, percept)
 goal ← FORMULATE-GOAL (KB)
 search-space ← FORMULATE-PROBLEM (KB, goal)
 plan ←SEARCH (search-space, goal)
 while (plan not empty) do
 action ← RECOMMENDATION (plan, KB)
 plan ← remainder(plan, KB)
 return action
```

Fig. 9. Algorithm with the basic operation of goal-based agents [38]

4) *Utility-based agents*

Utility-based agents are similar to goal-base agents, but they also have an utility measure used to evaluate the level of success when reached the goal. For example, an agent whose goal is to go from "City A" to "City B" can achieve this goal taking two to four hours to complete the route. By defining a measure of time efficiency, the agent will know that completing the route in less time is better. Fig. 10 illustrates the basic structure of such agents and Fig. 11 describes its basic operation.

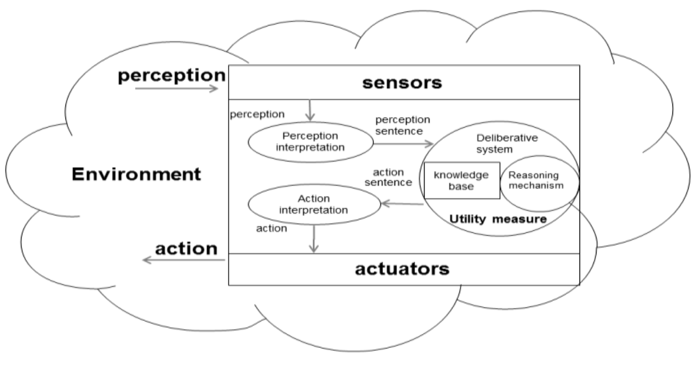

Fig. 10. The structure of an utility-based agent (adapted from [38])

```
function  Goal-Based-Agent (percept) returns action
static: KB, goal, search-space
KB ← UPDATE-KB (KB, percept)
goal ← FORMULATE-GOAL (KB, utility_measure)
search-space ← FORMULATE-PROBLEM (KB, goal)
plan ←SEARCH (search-space , goal)
while (plan not empty) do
action ← RECOMMENDATION(plan, KB)
plan ← REMAINDER (plan, KB)
return action
```

Fig. 11. An algorithm with the basic operation of an utility-based agents [38]

B. *Wooldridge Classification*

Wooldridge [19][21][22] classifies basic agents into three main categories: deductive reasoning agents, practical reasoning agents and reactive agents.

1) *Deductive reasoning agents*

Deductive reasoning agents have a symbolic model of their enviroment and their behavior is explicitly represented, typically using logic. The agent handles this representation by deductive reasoning which can require considerable time to be performed. Fig. 12 illustrates the structure of a deductive reasoning agente.

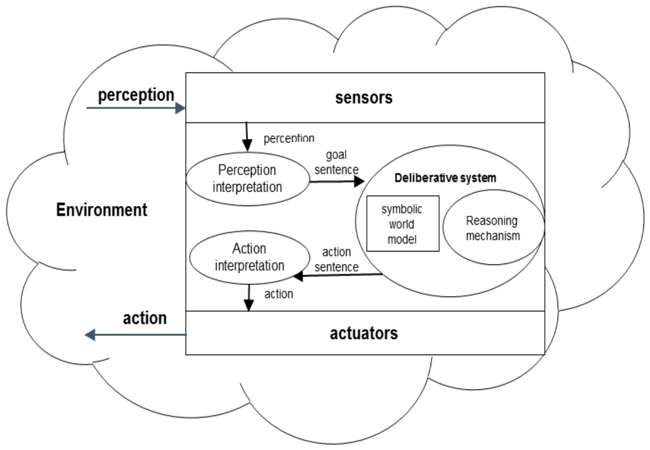

Fig. 12. The structure of a deductive reasoning agent (adapted from [21])

2) *Pratical reasoning agents*

Practical reasoning agents, also called BDI agents, are their based on the idea that agent acts not only by deductive reasoning. In practical reasoning, the decision process is performed based on beliefs, desires and intentions. Agent beliefs include the agent knowledge about the world, desires are some kind of desirable state to be reached, and intentions are the actions that the agent decides to take to achieve their desires.

To understand BDI agents, in [8] is given the following example: When a person graduates from university with a first degree, he/she is faced with some important choices. Typically, he/she proceeds in these choices by first deciding what sort of career to follow. For example, one might consider a career as an academic, or a career in industry. The process of deciding which career to aim for is deliberation. Once one has fixed upon a career, there are further choices to be made; in particular, how to bring about this career. Suppose that, after deliberation, you choose to pursue a career as an academic. The next step is to decide how to achieve this state of affairs. This process is means–ends reasoning. The end result of means–ends reasoning is a plan or recipe of some kind for achieving the chosen state of affairs. For the career example, a plan might involve first applying to an appropriate university for a PhD place, and so on. After obtaining a plan, an agent will typically then attempt to carry out (or execute) the plan, in order to bring about the chosen state of affairs. If all goes well (the plan is sound, and the agent's environment cooperates sufficiently), then after the plan has been executed, the chosen state of affairs will be achieved.

In Fig. 13 the main elements of a BDI architecture are illustrated. This architecture consists of a set of current beliefs that represent the information the agent has about its environment; by a belief revision function (brf - beliefs review function), that is the entry of a perception and current agent beliefs; a generation options function (generate options), which determines the choices of actions available to the agent (desires), based on their current beliefs about the environment and their current intentions; by a set of current options (desires), representing the actions available to the agent; by a filter function (filter) which represents the agent's deliberation process and determining the intentions of the agent based on their current beliefs, desires and intentions; by a set of current intentions (intentions), representing the current goal of the agent and by a selection function action (action) that determines an action to be performed on the basis of current intentions.

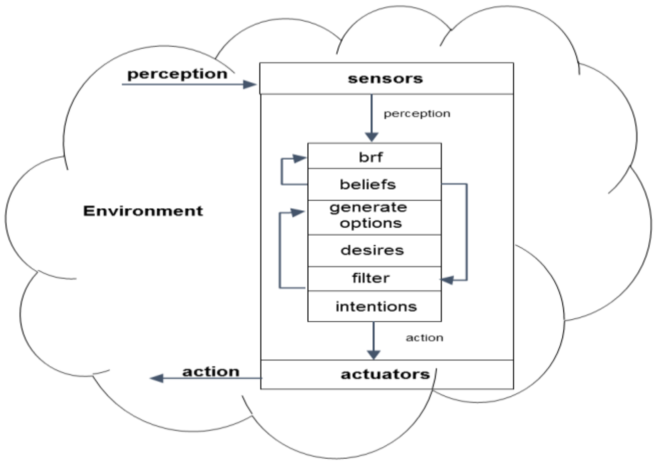

Fig. 13. The structure of a BDI agent (adapted from [21])

The main advantages of BDI architectures, cited in [21], are conceptual proximity to the process of human decision and the informal understanding of the notions of belief, desire and intentions.

3) *Reactive agents*

Reactive architectures are defined with the aim of filling the main shortcoming of logic-based architectures: processing time. The goal of the architecture is reactive agent that can provide intelligent behavior through a set of quick and simple behaviors. The knowledge of the agent and its behavior is not necessarily represented in logic and the agent does not perform any kind of reasoning.

In Wooldridge reactive architecture (illustrated in Fig. 14) is defined a set of rules for direct mapping of perceptions to actions. Some reactive architectures are organized into layers with different levels of abstraction, where the lower layers have a higher level of priority, i.e., critical actions are performed by these layers. The layers can also be independent, i.e., each layer can process a perception and perform an action. In this case, the actions can be executed in parallel.

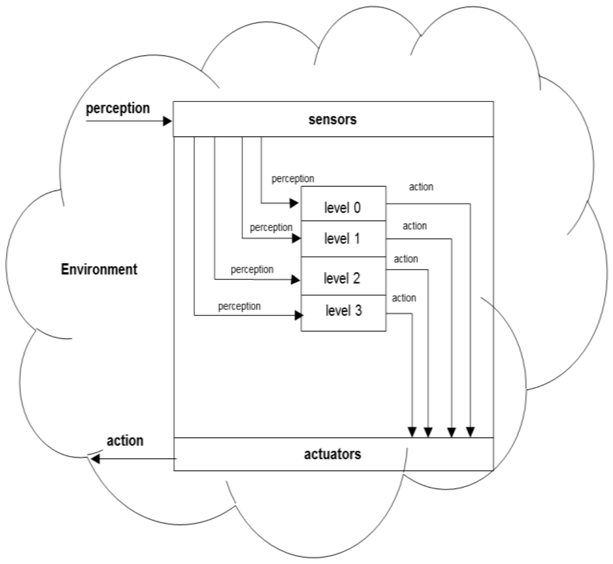

Fig. 14. The structure of a reactive agent (adapted from [21])

C. *Kendall Classification*

Kendall [3] defined two agent architectures organized into layers: layered agents and reactive agents. A difference this author to others is that it represents the agents through patterns.

1) *Reactive agents*

Kendall defines the following pattern for structuring a reactive agent (Fig. 15).

Problem: How can an agent react to an environmental stimulus or a request from another agent when there is no symbolic representation and no known solution?

Forces: An agent needs to be able to respond to a stimulus or a request; there may not be a symbolic representation for an application and an application may not have a knowledge based, prescriptive solution.

Solution: A reactive agent does not have any internal symbolic models of its environment; it acts using a stimulus/ response type of behavior. It gathers sensory input, but its belief and reasoning layers are reduced to a set of situated action rules. A single reactive agent is not proactive, but a society of these agents can exhibit such behavior.

Known Uses: Reactive agents have been widely used to simulate the behavior of ant societies and to utilize such societies for search and optimization.

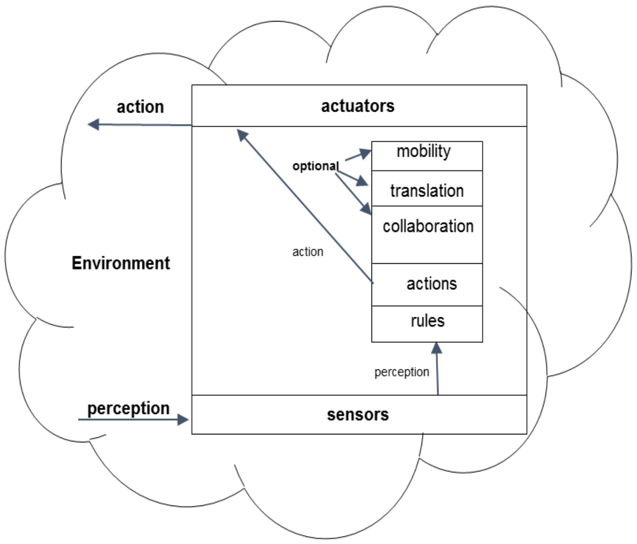

Fig. 15. The reactive agent of the Kendall architecture (adapted from [3])

2) *Layered agents*

Kendall specified the following pattern for layered agent (illustrated in Fig. 16):

Problem: How can agent behavior be best organized and structured into software? What software architecture best supports the behavior of agents?

Forces: An agent system is complex and spans several levels of abstraction; there are dependencies between neighboring levels, with two way information flow; the software architecture must encompass all aspects of agency; the architecture must be able to address simple and sophisticated agent behavior.

Solution: Agents should be decomposed into layers because i) higher level or more sophisticated behavior depends on lower level capabilities, ii) layers only depend on their neighbors, and iii) there is two way information flow between neighboring layers. The architecture structures an agent into seven layers. The exact number of layers may vary. In the sensory layer the agent perceives its environment through sensors. Agents beliefs are based on sensory input, so, in the beliefs layer, perceptions are mapped to logic sentences that are included in the agent knowledge base, which is part of this layer and where the agent maintains models of its environment and itself. In the Reasoning layer, when presented with a problem, the Agent reasons about the symbolic model in the knowledge base to determine what to do by selecting a particular action to perform on the environment. An agent selects a plan to achieve a goal. When the agent decides on an action, it can carry it out directly, but an action that involves other agents requires collaboration. Once the approach to collaboration is determined, the actual message is formulated and eventually translated into other semantic and delivered to distant societies by mobility. Top-down, distant messages arrive by mobility. An incoming message is translated into the agent's semantics, The collaboration layer determines whether or not the agent should process a message. If the message should be processed, it is passed on to actions. When an action is selected for processing, it is passed to the reasoning layer, if necessary. Once a plan placed in the actions layer, it does not require the services of any lower layers, but it utilize higher ones.

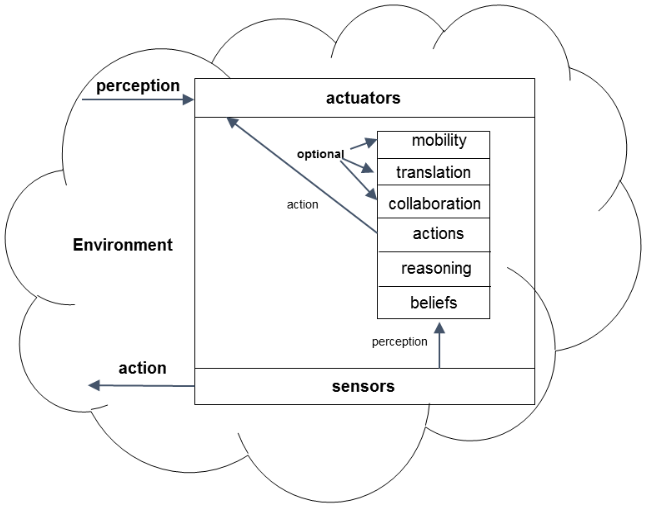

Fig. 16. Layered agent (adapted from [3])

3) *A summary of basic agent architectures*

Table I summarizes different main architecture the architectures of Kendall's discussed in this section.

The simple reflex agent and agent with states defined by Russel and Norvig correspond to Wooldridge reactive agent and the Kendall reactive agent. A main difference is that Wooldridge and Kendall defines that architectures can be arranged in layers with increasing level of abstraction.

The goal-based and utility-based agents defined by Russel and Norvig correspond to the Wooldridge deductive reasoning agent and Kendall layered agent. Russell and Norvig defines a measure of performance and utility. Kendall organizes their deliberative architecture in layers. In general, these architectures are called deliberative architectures and use some kind of reasoning.

Wooldridge defines an architecture called practical reasoning, also neither known as BDI agent, whose main concepts are not approached by Kendall and Russell and Norvig.

TABLE I . CORRESPONDING TERMINOLOGY OF AGENT ARCHITECTURES

Kind of agent	Russel and Norvig	Kendall	Wooldridge
Reactive	Reflex agent	Reactive	Reactive Agent
	Reflex agent with state	Agent	
Deliberative	Goal-based agent	Layered Agent	Deductive Reasoning Agents
	Utility-based agent		Practical Reasoning Agents

III. BASIC AGENT COMPONENTS

An agent has a set of components that are part of its internal architecture, also called agent structure. These components vary according to the agent type.

In this section, basic components of a software agent as reasoning, knowledge base and communication are presented.

A. *The Reasoning Component*

Reasoning is the process of making inferences about a set of assumptions in order to obtain conclusions. There are four main types of reasoning: deduction, induction, abduction and analogy. Deliberative software agents have mechanisms of reasoning and use it to perform more complex actions than reactive ones.

Deduction is the most rigorous kind of reasoning. When deduction premises are true, necessarily also be the conclusion will be also free. For example: "All men are mortal" (premise 1), "Socrates is a man" (premise 2), "Therefore, Aristotle is mortal" (conclusion).

Inductive reasoning part from the observation of objects and of the similarity between its properties. From the observation of the common features of a limited set of objects, an entire category is generalized. For example: "Canaries flies" (Premise 1), "Parrots flies" (Premise 2), "Pigeons flies" (Premise n), "therefore, all birds fly" (Conclusion). Conclusion is not universally true, just likely to be true.

Abductive Reasoning [30] is also called inference by the best explanation. In abduction, as in induction, the conclusion is not an universal truth, but has the probability of being true. Abduction prepares explanatory hypotheses for these observations and hypotheses are evaluated. For example, given the observation that "The street is wet", the following hypothesis can be formulated: "It rained " (hypothesis 1) and "A water truck started pouring water" (hypothesis 2). But when a new observation like "The house roof is wet." Is obtained, the hypothesis 2 should be rejected and hypothesis 1 validated. Abductive reasoning is often used by criminologists, detectives and diagnosis of diseases.

There is another reasoning process called reasoning by analogy, which goes from the particular to the particular. This process occurs with the perception and classification of objects according to similar features and deriving a conclusion from a previous experience in one or more similar situations [18]. For example, consider the set of individuals A, B and C, having the following range of characteristics: white color, thin, blond hair and blue eyes. Given that the individual A has the high stature characteristic, it should be possible to conclude that B and C are likely to have this feature even if not explicit in the definition.

Analogy is a type of reasoning often used in everyday life and has already been successfully applied in various fields of knowledge. For example, the first plans for building vehicles very similar to current helicopters made by Leonardo da Vinci mimicked the mechanisms used by birds to fly [24].

B. The Knowledge Base Component

Another component of a deliberative agent architecture is its knowledge base. The agent knowledge base contains the knowledge of the external environment, the agent perception history and the knowledge of rules for mapping perceptions to actions.

The most common way to represent agent knowledge is symbolic representation, using logic programming languages, like Prolog [17] and Jess [4] and knowledge representation languages supporting semantic networks, frames or ontologies, like RDF [27] and OWL [2]. Ontologies are knowledge representation structures capable of expressing a set of entities in a given domain, their relationships and axioms, being used by modern knowledge-based systems as knowledge bases to represent and share knowledge of a particular application domain. They allow semantic processing of information and a more precise interpretation of data, providing greater effectiveness and usability than traditional information systems [32].

Before being used by an inference engine, perceptions must be transformed into sentences in a knowledge representation language understandable by the agent. After finding, through reasoning, an action representing an appropriate solution for the perception, this action should be interpreted and performed on the environment. For example, in the case of the vacuum cleaner, a sentence that represents the action of moving forward, should rather be interpreted and mapped prior to electronic signals that indicate to the hardware of the vacuum cleaner that he should move. When the agent environment is artificial, like the Internet, the agent perceptions can be just text in natural language or events like a mouse click. Actions can just display text on the screen, send a message or trigger an event like a sound alarm.

It should be distinguished between the agent internal knowledge and the knowledge shared with other agents of the society and / or external entities. The agent internal knowledge is just necessary for performing its own actions. Shared knowledge is required for agents to communicate.

C. The Communication Component

Software agents usually specialize in just certain tasks. Thus, frequently, agents need to communicate to accomplish tasks that are beyond their individual capabilities in order to achieve the overall goal of the multi-agent system. Thus, in these systems, agents need to communicate. Communication among agents is based on the theory of speech acts [37] and for that, Agent Communication Languages (ACLs) have been developed, like KQML (Knowledge Query and Manipulation Language)[7][41] and FIPA-ACL (Foundation for Intelligent Physical Agents - Agent Communication Language) [16] for expressing communication acts and supporting the coordination and cooperation mechanisms of a multi-agent society architecture.

. In the theory of speech acts, a message intent is called performative. For example, the intention of an agent may be a request such as "Can you send me the price of the blue blouse?" or just information like "The summer time in Brazil began on October 20, 2013." According to their main intentions a speech act can inform, question, answer, ask, offer, confirm and share.

IV. ADVANCED AGENT ARCHITECTURES

Basic agent architectures presented in the previous sections already have a high level of maturity, having techniques [11][13][33][34] and frameworks [6][14][15] that support their development. More advanced agent architectures such as learning and hybrid ones are still an open research topic. This section discusses these more advanced software agent architectures.

A. Learning Agent Architectures

The idea behind learning is that perceptions should be used not only to act but also to improve the agent ability to act in the future [38].

Basic software agents have no learning; they act according to the perceptions defined in the agent design. Therefore, for new perceptions the agent must be reprogrammed.

Learning agents can at runtime change their behavior according to changes in the environment. In this type of agents, perceptions should be used not only to act but also to improve the agent ability to act in the future.

According to Russell and Norvig [38], a learning agent has four basic components (Fig. 17): performance, critic, learning and problem generator.

The performance component is what we have previously considered to be a basic agent: perceives and acts on the environment

The learning component is responsible for making the agent behavior improvements. It uses feedback from the critic on how the agent is doing and determines how the performance component should be modified to do better in the future. The critic tells the learning component about the success of the agent according to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent success.

The last component of the learning agent is the problem generator which is responsib1e for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. The problem generator main goal is to suggest these exploratory actions. This is what scientists do when they carry out experiments.

An example of the functioning of a learning agent architecture, the automated taxi, is given in [38]. The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions. The critic observes the world and passes information along to the learning element. For example, after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by others drivers. From experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installation of the new rule. The problem generator might identify certain areas of behavior in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

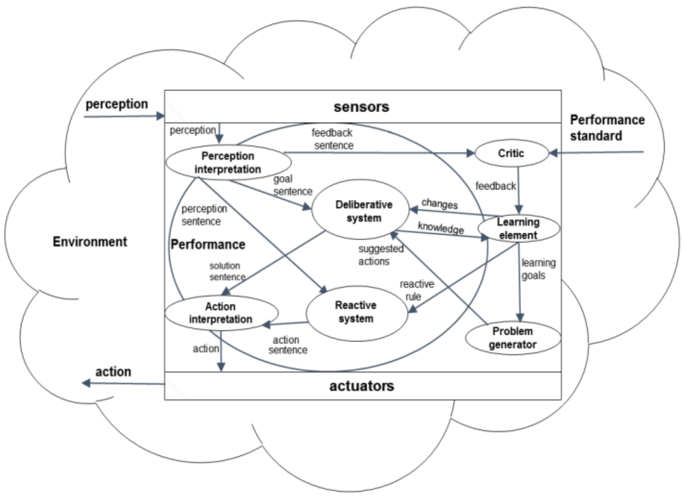

Fig. 17. The structure of a learning agent (adapted from [38])

During the construction of the learning element it is necessary to define the learning technique to be used. Well-known techniques of machine learning are supervised learning, unsupervised learning and reinforcement learning [36].

The problem of supervised learning involves learning a function from a set of inputs and outputs examples which will be later used to produce the correct output given a new input. Examples of these techniques are decision trees, Bayesian networks and neural networks [38]. Supervised learning is associated with two common problems which are classification and regression. Classification is to assign an instance to a class through a classifier previously built. A classifier can be, for example, a decision tree or set of rules. Regression attempts to identify a output and represent it by a numeric value from a set of training data.

A disadvantage of a supervised learning technique is that learning is dependent on the training examples and creating these training examples may require considerable time and effort.

The problem of unsupervised learning involves learning patterns in the input and building a model or useful representations of the data, for example, clusters when no specific output values are supplied. For example, a taxi agent might gradually develop a concept of “good traffic days” and “bad traffic days” without ever being given labelled examples of each [38]. This could be done through the development of clusters.

A cluster is a collection of objects that are similar to each other (according to some pre-defined similarity criterion) and not similar to objects belonging to other clusters. So it could be constructed clusters representing "good traffic days" and "bad traffic days". These clusters could have characteristic values as "time spent route" and "fuel cost". After clusters construction, the agent could learn which days probably has good traffic. Clusters do not always have adequate exits after completion of the training dataset. When this happens, it is necessary to evaluate why the output is not appropriate. Some data output is not correct because there were used insufficient training examples or there were defined non-relevant features for the objects that compose the clusters.

Reinforcement learning is inspired in the behaviorist psychology where an agent learns to act in a way that maximizes rewards in the long term. Reinforcements are obtained through the interaction of the agent with the environment and can be positive (reward) or negative (punishment). In reinforcement learning there is no examples of correct output. The reinforcement obtained through interaction with the environment is used to assess the agent behavior and is associated with a performance standard establishing whether that reinforcement is positive or negative. Better agent performance is obtained through experience.

A reinforcement learning example is of a mouse that moves about a maze trying to get the cheese while avoiding the deadly trap. The mouse does not know beforehand the layout of the maze or the placing of the cheese/trap. At each square, it must choose whether to move up, down, left or right. The mouse will explore the maze to find the cheese. After finding the cheese, it will already know which path took to find the cheese (positive reinforcement), but it still tries to find a shorter way to the cheese (to maximize its performance measure). After various experiences in the maze it will learn the shortest path from the starting point until the cheese.

B. Hybrid Agent Architectures

Hybrid architectures, also known as layered architectures have emerged from the need to gather in a single agent reactive and deliberative behavior. Wooldridge classifies this type of architecture into two groups [21]: hybrid architectures in horizontal layers and hybrid architectures in vertical layers.

In horizontal layers hybrid architectures (Fig. 18), each software layer is connected directly to a sensor and an actuator. In this type of architecture each layer works as an independent agent. One advantage of organizing horizontally layered architectures is the clear separation of the different agent behaviors within the architecture, where each layer can act independently of the other, even in parallel. One of the problems with this architecture type is that the overall behavior of the agent cannot be consistent. To solve this, usually a control layer is also designed to ensure a coherent overall behavior.

In vertical layers hybrid agent architectures, perceptions and actions are handled by more than one layer. In these architectures, layers can be further subdivided into a single-passage (Fig. 19) and two-passages (Fig. 20) architectures. In the architecture of a single-passage, the control flow passes sequentially through each layer until reaching the final layer where the action to be performed in the environment is generated. In the two-passages architecture, information flows to reach the final layer (first pass) and control then flows back down (second pass).

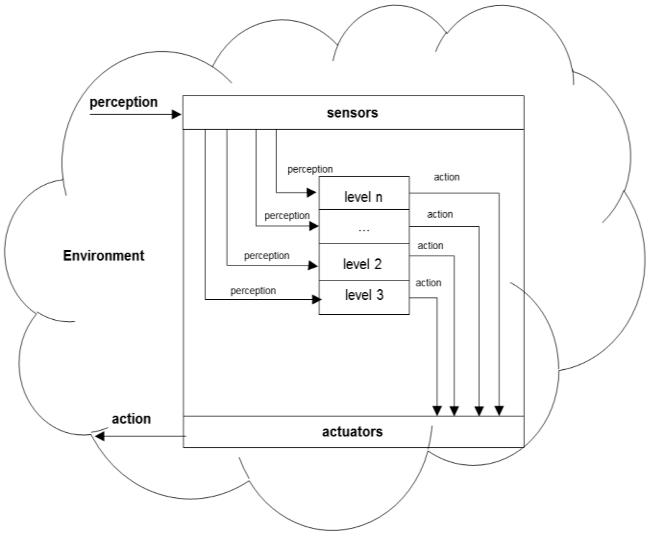

Fig. 18. A horizontal layered agent architecture (adapted from [21])

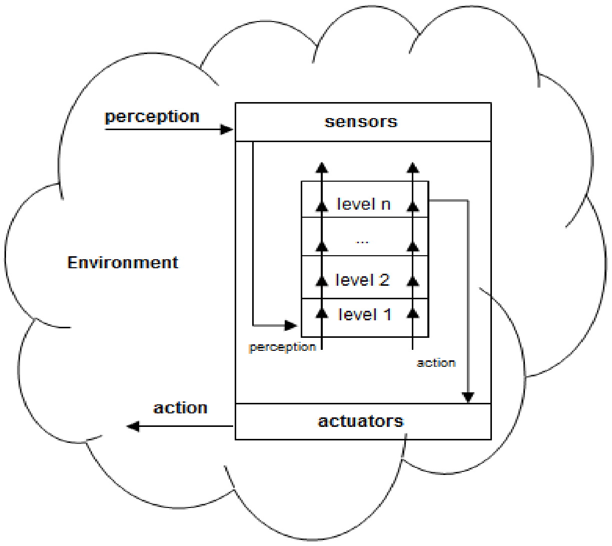

Fig. 19. A vertical layered agent architecture of a single-passage (adapted from [21])

Russell and Norvig [38] also propose a hybrid architecture (Fig. 21). The agent architecture has two main components they called subsystems: a deliberative and a reflex systems. The goal of the architecture is to exhibit more efficient agent behavior by converting deliberative decisions into reflective ones, making the agent actions more fast and efficient.

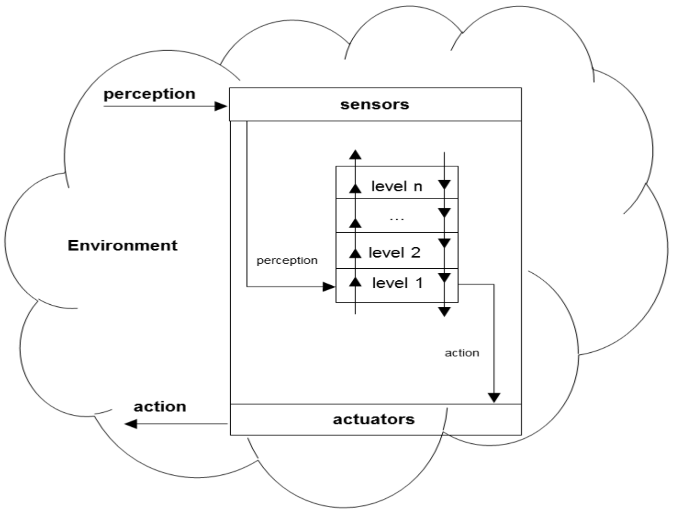

Fig. 20. A vertical layered agent architecture of two-passages (adapted from [21])

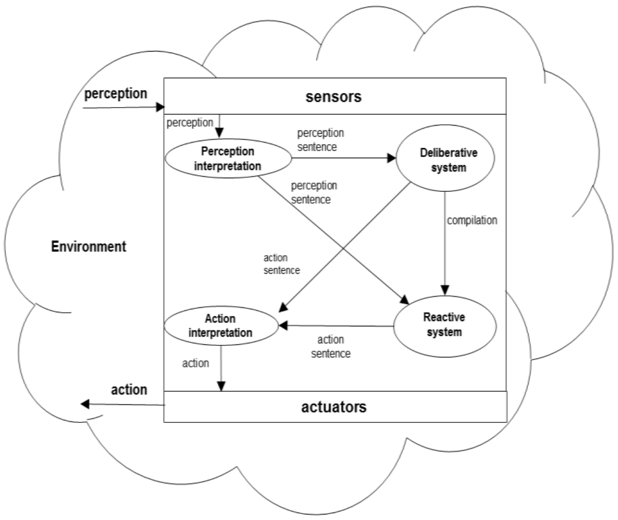

Fig. 21. Another proposal of a hybrid agent architecture (adapted from [38])

Examples of current hybrid architectures are SOAR [5][10], ACT-R (Adaptive Control of Thought–Rational)[9], INTERRAP (Integration of Reactive Behavior and Rational Planning) [29]. A common feature of most proposal of current hybrid architectures is that they are cognitive. A cognitive architecture aims at developing agents capable of acting using human cognitive phenomena such as memory, learning, decision making and natural language processing [28].

SOAR was one of the first proposals of hybrid architectures. It has a development environment and a framework to support the creation of agents according to their definitions. It is also a goal-driven cognitive architecture that integrates reasoning, reactive execution, planning and various learning techniques, aiming at creating a software system having similar cognitive abilities as humans. The SOAR agent architecture is illustrated in Fig. 22. This architecture is composed of a working memory, also known as short term memory, a long-term memory, a reasoning module, modules responsible for managing the agent perceptions and actions and learning modules.

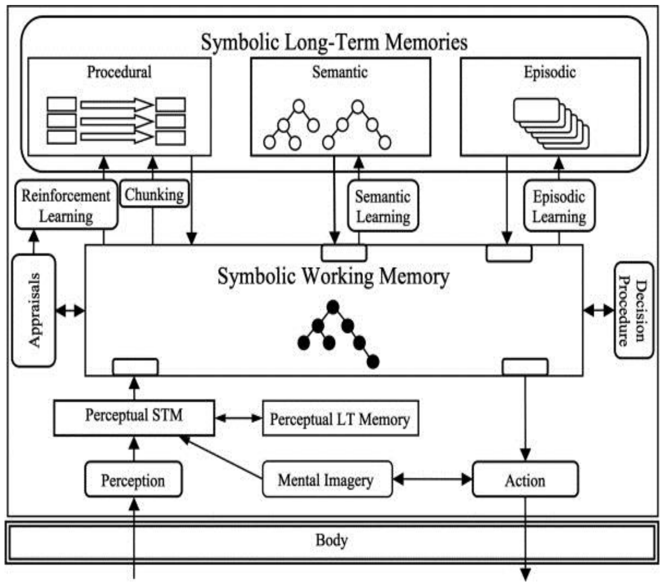

Fig. 22. The SOAR hybrid agent architecture [5]

ACT-R is a programming environment that supports the development of hybrid agents. The ACT-R hybrid architecture (Fig. 23) attempts to emulate cognitive processes of human cognition such as knowledge acquisition and learning through a production system. It consists of a perceptual/motor layer, a cognition layer and a buffer intermediate layer. In the perceptual/motor layer, input and output information processing is emulated through the human visual, motor, speech and hearing modules. In the cognition layer, the memory of the agent is represented by declarative and production modules. The declarative memory corresponds to the reactive part of the system and consists of "chunks". A chunk is an attribute-value structure; with a special type of attribute called "ISA" that determines the type of the chunk. The production memory is formed by condition-action rules and an inference engine to select actions in the knowledge base. The representation of the two types of memories for the agent is what makes ACT-R a hybrid architecture. The intermediate module ACT-R buffers is the working memory of the agent. It corresponds to the knowledge used only when performing a particular task. This knowledge can be retrieved both by the declarative memory and the production memory. Learning in the ACT-R architecture is by "chunking" which basically consists in useful pieces of information (chunks) that are stored in the declarative memory for future use.

The InteRRaP architecture (Fig. 24) is a two-passages hybrid architecture that supports the development of reactive agents and goal-based agents. It is organized into layers together with a control structure and a knowledge base associated with each layer. It consists of five main components: an interface with the world (WIF), a component-based behavior (BBC), a plan-based component (PBC), a cooperation component (CC) and the agent knowledge base. The WIF component enables perception, action and agent communication. The BBC component supports reactive behavior and represents procedural knowledge. The PBC component contains planning mechanisms for constructing agent plans. The CC component contains a mechanism to compose these plans. The knowledge base of InteRRaP consists of three layers. The lowest layer contains facts that represent a model of the agent environment as well as representations of actions and behavior patterns. The second layer contains the agent mental model. The third layer consists of the agent social model, which provides strategies for cooperation with other agents.

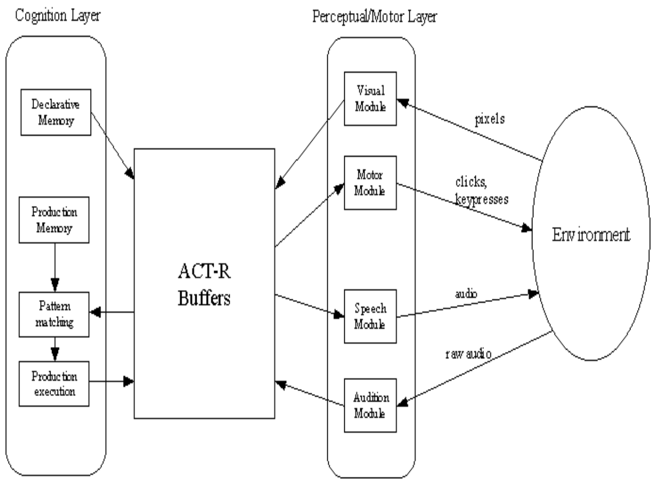

Fig. 23. The ACT-R hybrid agent architecture [26]

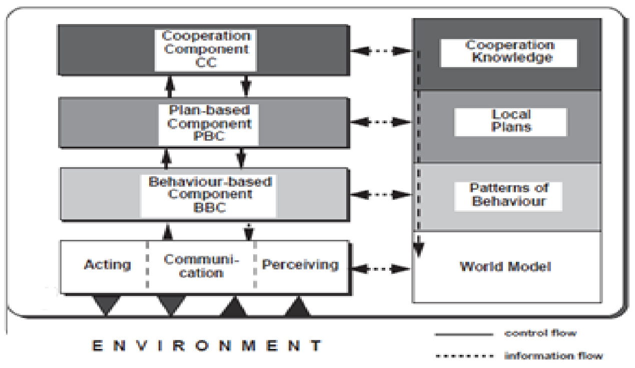

Fig. 24. The InteRRaP hybrid agent architecture [29]

Qinzhou and Lei [12] define a hybrid agent architecture using case-based reasoning and unsupervised learning (Fig. 25) composed of eight modules: a knowledge module in charge of storing a set of cases and a set of rules; a module responsible for the condition-action rules corresponding to the reactive behavior; a perception module responsible for receiving information from the environment, a learning module that uses unsupervised learning algorithms to increase the efficiency of case retrieval; a retrieval module whose main function is to compare and perform the similarity computation between a new case with an old case in the case base; a decision module which corresponds to the deliberative behavior, responsible for performing a process of reasoning on cases using the set of rules from the knowledge module; an execution module responsible for performing actions on the environment; and a communication module responsible for the interactions between agents, which uses the KQML[7].

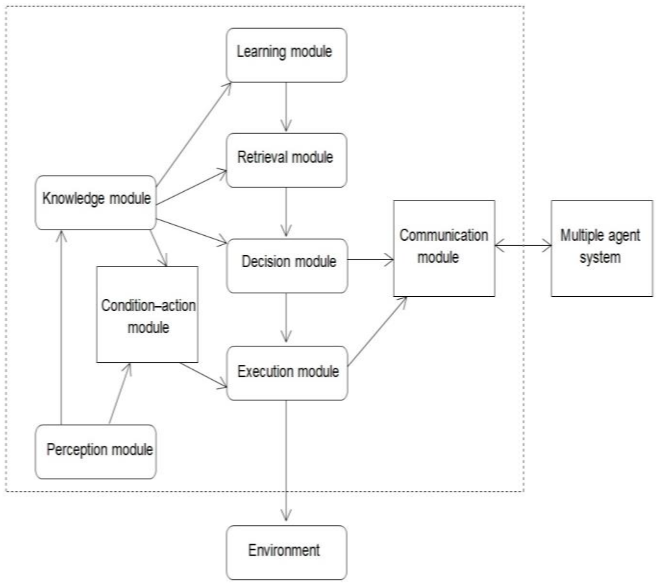

Fig. 25. The Qinzhou and Lei hybrid agent architecture [25]

Table II shows a comparison between the SOAR, ACT-R, INTERRAP and Qinzhou and Lei hybrid architectures by considering the characteristics of their reactive and deliberative components, the applied learning technique and the representation mechanisms of the knowledge base.

Among these architectures, and probably among all hybrid architectures of the state of the art, the broader one is the SOAR architecture which includes various forms of learning and mechanisms for representing the agent knowledge base. SOAR also provides tools for developing agents, updated documentation and examples of implemented agents for various problems.

Table II. A Comparison of Hybrid Agent Architectures

Features / **Architecture**	**Reactive Component**	**Deliberative Component**	**Learning Technique**	**Knowledge base representation**
SOAR	Reactive with state	Deductive	Reinforcement, episodic, chunking and semantic	Condition-action rules, state graphs and semantic memory.
ACT-R	Reactive with state	Deductive	Chunking	Rules, facts and procedural knowledge
INTERRAP	Reactive with state	BDI	No learning	Procedural knowledge
Qinzhou and Lei Architecture	Reactive simple	RBC	Unsupervised learning	Cases

The ACT-R architecture differs from the SOAR architecture because it is more restricted in relation to the alternative representation of the agent knowledge base. However, this architecture has a wider treatment of perceptions, including representing images and voice.

INTERRAP's main advantage over other hybrid architectures presented in this section, is its layered organization, considering that the definition of several independent components inserts complexity in the architecture design, making necessary to manage the interactions between the components.

The architecture of Qinzhou and Lei differs from the others by using case-based reasoning and unsupervised learning to classify similar cases in a class according to the most relevant characteristics of the cases.

V. CASE STUDY

OHAA ("Ontology-driven Hybrid Agent Architectures") is an ontology-driven hybrid and learning agent architecture that combine deliberative and reactive components joining the advantages of both behaviors to improve the decision making process. Thus, the agent may have a reactive behavior or a deliberative one depending on its perception and available behavior.

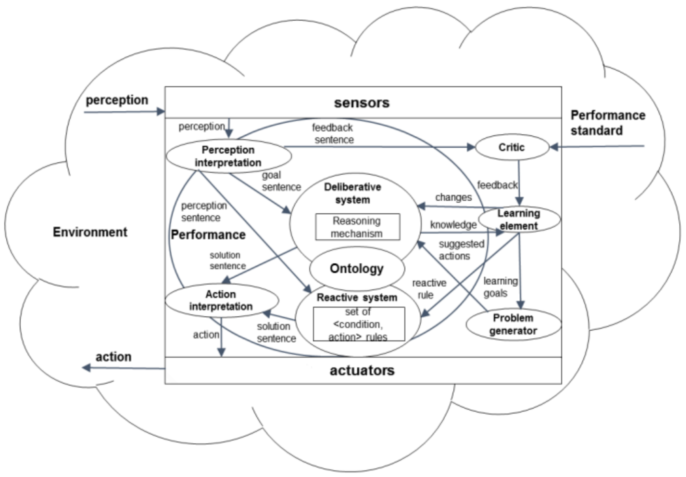

Fig. 26. The structure of OHAA Architecture

Additionally, the architecture allows learning new reactive rules through recurrent solutions to the same perception from the deliberative system, which will be stored in the agent knowledge base. Also, the learning component supports the evolution of deliberative to reactive behavior. Finally, in OHAA, the knowledge base is represented as an ontology thus enabling knowledge improvement through reuse. Fig. 26 illustrates the OHAA basic structure.

A first approach of the OHAA functioning is described as follows.

1. Interpreting the perception;
2. Mapping the perception to a sentence;
3. Asserting the perception sentence in the knowledge base ontology;
4. If there is a rule for the perception, the corresponding

action to perform is selected by the reactive system;

5. If there is no a reflex action corresponding to the perception, this will be treated by the deliberative system that will reason to find the most appropriate action;
6. Upon completion of the action, the agent will perceive a feedback from the environment about the success or failure of the performed action;
7. In the critical component, which is fed by a performance standard of the agent actions from the environment, the feedback perception will be evaluated. If the action was poorly evaluated by the critical, this component informs the learning component;
8. If the action was assessed as good, this behavior will remain in the knowledge base;
9. Following, the learning component makes recommendations for improvements in the actions of the agent;
10. These recommendations are passed to the problem generator component which, in turn can generate a new set of possible actions for the agent;
11. When the agent performs these new actions, suggested by the problem generator component, it will have a feedback perception and the process restarts;
12. Finally, actions repeatedly well evaluated will be transformed into reactive rules;
13. When the agent performs these new actions, suggested by the problem generator component, it will have a feedback perception and the process restarts;
14. Finally, actions repeatedly well evaluated will be transformed by the learning component into reactive rules.

A. *A simples example*

Consider a genealogy tree as the OHAA environment (Fig. 27). By traversing the tree the agent just perceives who are the parents of a given person.

OHAA also has knowledge about genealogy (Fig. 28) so it can conclude through deductive reasoning who are the kins of a given person. For example, through the knowledge base inference rules of the deliberative system of Fig. 28, the agent can conclude that Bob and Julie are cousins.

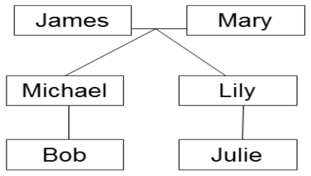

Fig. 27. Genealogy Environment

If the action generated through the inference that conclude that "Bob and Julie are cousins" is repeatedly well evaluated then it could be transformed by the learning component into a reactive rule. When the rule becomes reactive, the agent does not need to reason once the environment is static. Then, the knowledge base of the agent will be updated with the information that "Bob and Julie are cousins". In the knowledge base (Fig. 29) this reactive rule is represented by the "cousins (bob, julie)." fact.

```
Deliberative system
parent(X,Y) :- father(X,Y).
parent(X, Y) :- mother(X,Y).
brothers(X,Y):- parent(X,Z),parent(Y,Z).
cousins(X,Y):-brothers(A,B),parent(X,A),parent(Y,B).

Reactive system
father(Michael,James).
father(Lily,James).
father(Bob,Michael).
mother(Michael,Mary).
mother(Lily,Mary).
mother(Julie,Lily).
```

Fig. 28. Facts and inference rules in the reactive and deliberative systems of the OHAA knowledge base

```
Deliberative system
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
brothers(X,Y) :- parent(X,Z),parent(Y,Z).
cousins(X,Y) :- brothers(A,B),parent(X,A),parent(Y,B).

Reactive system
father(michael,james).
father(lily,james).
father(bob,michael).
mother(michael,mary).
mother(lily,mary).
mother(julie,lily).
cousins(bob,julie).
```

Fig. 29. The representation of a learned reactive rule in the OHAA knowledge base

VI. CONCLUDING REMARKS

This paper presented a study about basic and advanced software agent architectures. The main features of reactive and deliberative agent architectures, their internal components and how they relate to each other were described. Hybrid architectures which combine both reactive and deliberative agent behavior and learning architectures allowing the improvement of the agent behavior were also analyzed.

Considering that they simulate better intelligent human behavior, current work focuses on the design of hybrid and learning architectures, providing frameworks and design tools for agent construction.

Additionally, the OHAA hybrid architecture has also been introduced. The OHAA Architecture combines deliberative and reactive components joining the advantages of both behaviors to improve the decision making process. A learning component also was defined in OHAA, responsible for learning new agent behaviors and for transforming deliberative behaviors into reactive ones.

An example of OHAA utilization and a comparative study of main approaches of hybrid agent architectures have been also discussed.

The hybrid architecture OHAA is still in an early stage. Current work looks for detailing the architecture components and evaluating its effectiveness through the design and implementation of an initial prototype and the development of a case study in the family law legal field using case-based reasoning [1] and instance-based learning [24].

Further work will specify a technique and implementing a tool for constructing agents using the OHAA architecture. More evaluation experiments will be conducted using deductive reasoning and reinforcement learning [40].

REFERENCES

[1] A. Aamodt, E. Plaza, "Case-Based Reasoning: Foundation Issues, Methodological Variations, and System Approaches", *AICOM*, Vol. 7: 1,1994, pp. 39-59.

[2] B Motik, "OWL 2 web ontology language: Structural specification and functional-style syntax", *W3C recommendation*, v. 27, 2009

[3] E. A. Kendall, P. V. Krishna, C.V. Pathak, C. B. Suresh, "Patterns of intelligent and mobile agents". In *Proceedings of the second international conference on Autonomous agents*, 1998, pp. 92-99.

[4] E. Friedman-Hill, "*JESS in Action. Greenwich*", CT: Manning, 2003.

[5] E. Laird, "*The Soar cognitive architecture*", MIT Press, 2012.

[6] F. Bellifemine, G. Caire, D. Greenwood, "*Developing Multi-Agent Systems with JADE*", Wiley, 2007, ISBN: 978-0-470-05747-6.

[7] F. Tim, Y. Labrou, J. Mayfield, "*KQML as an agent communication language*". Computer Science Department. University of Maryland Baltimore County. Baltimore, USA, 1993.

[8] G. Weiss, "Multiagent Systems Intelligent" In: *Robotics and Autonomous Agents series*, The MIT Press, 2013.

[9] J. Ball, "Explorations in ACT-R Based Cognitive Modeling-Chunks, Inheritance, Production Matching and Memory in Language Analysis," In: *Proceedings of the AAAI Fall Symposium: Advances in Cognitive Systems*, 2011, pp. 10-17.

[10] J. Laird, "*The Soar 9 Tutorial*", University of Michigan, 2012.

[11] J. Mylopoulos, J. Castro, M. Kolp, "The Evolution of Tropos". In: *Seminal Contributions to Information Systems Engineering*. Springer Berlin Heidelberg, 2013, p. 281-287.

[12] K. Riesbeck, C. Schank, "*Inside Case-based Reasoning*", Psychology Press, 2013.

[13] K. Vivekanandan, D. RAMA, "Analysing the Scope for Testing in PASSI Methodology". *International Journal*, v. 3, n. 1, 2013.

[14] L. Braubach, A. Pokahr, "Jadex Active Components Framework-BDI Agents for Disaster Rescue Coordination", In: *Software Agents, Agent Systems and Their Applications,volume 32 of NATO Science for Peace and Security Series*, IOS Press, 2012.

[15] L. Braubach, A., Pokahr, "Developing Distributed Systems with Active Components and Jadex", *Scalable Computing: Practice and Experience*, v. 13, n. 2, 2012.

[16] L. Shujun, M. Kokar, "*Agent Communication Language. Flexible Adaptation in Cognitive Radios*", Springer New York, 2013, pp. 37-44.

[17] L. Sterling, E. Shapiro, "*The Art of Prolog: Advanced Programming Techniques*", 2nd Edition, MIT Press, 1994, 688 pages, ISBN 0-262-19338-8.

[18] L.W. Lloyd. "*Legal Reason: The Use of Analogy in Legal Argument*", New York: Cambridge University Press, 2005.

[19] M. Wooldridge, P. Ciancarini. "Agent- oriented Software Engineering: The State of the Art", In: *"Agent-Oriented Software Engineering"*, Springer-Verlag, Lecture Notes in AI Volume 1957, 2000.

[20] M. Shaw, Garlan, D. "*Software architecture: perspectives on an emerging discipline*", Prentice Hall, 1996.

[21] M. Wooldridge, "*An Introduction to Multiagent Systems*", 2nd ed.,Wiley Publishing, 2009.

[22] M. Wooldridge, "Intelligent Agents", In: *"Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence"*, G. Weiss (ed.), The MIT Press, 1999.

[23] N. R. Jennings, "On Agent-based Software Engineering", *Artificial Intelligence Journal*, 2000, pp. 277-296.

[24] N. Jiang et al., "Application of bionic design in product form design", In: *Computer-Aided Industrial Design & Conceptual Design (CAIDCD)*, 2010, pp. 431-434.

[25] N. Qinzhou, H. Lei, "Design of case-based hybrid agent structure for machine tools of intelligent design system", In: *2012 Software Engineering and Service Science (ICSESS 2012)*, pp.59-62, 2012.

[26] N. Taatgen, A. Niels, C. Lebiere, J. R. Anderson, "Modeling paradigms in ACT-R", *Cognition and multi-agent interaction: From cognitive modeling to social simulation*, 2006, pp. 29-52.

[27] O. Lassila, R. Swick, "RDF/XML syntax specification". *W3C Recommendation*, v. 22, 2010.

[28] P. Langley, J. Laird, J. Rogers, "Cognitive architectures: Research issues and challenges", In: *Cognitive Systems Research*, v. 10, n. 2, p. 141-160, 2009.

[29] P. Müller, M. Pischel, "*The agent architecture InteRRaP: Concept and application*" 2011.

[30] P. Thagard, E.C. SHELLEY, "Abductive Reasoning: Logic, Visual Thinking, and Coherence", In: *M.L*, 1997.

[31] R. Girardi, "An Analysis of the Contributions of the Agent Paradigm for the Development of Complex Systems", In: *Joint meeting of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th International Conference on Information Systems Analysis and Synthesis (ISAS 2001)*, 2001, pp. 388-393.

[32] R. Girardi, "Guiding Ontology Learning and Population by Knowledge System Goals", In: *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, Ed. INSTIIC, Valence, 2010, p. 480 – 484.

[33] R. Girardi, A. Leite, "Knowledge Engineering Support for Agent-Oriented Software Reuse", In: *M. Ramachandran. (Org.). Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*. Hershey: IGI Global, v. I, p. 177-195, 2011.

[34] R. Girardi, A. Leite, "The Specification of Requirements in the MADAE-Pro Software Process", *iSys: Revista Brasileira de Sistemas de Informação*, v. 3, p. 3, 2010.

[35] R. Johnson, R. Helm, J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*". Addison-Wesley Professional, 1994.

[36] R. Sutton, R. Richard, A. Barto, G. Andrew, *"Reinforcement learning: An introduction"*, Cambridge: MIT press, 1998.

[37] S. Kuroda, "*A formal theory of speech acts. Linguistics and philosophy*", v. 9, n. 4, p. 495-524, 1986.

[38] S. Russel, P. Norvig, "*Artificial Intelligence: A Modern Approach* ", 3nd ed., Prentice-Hall, 2009.

[39] T. Gruber, "Toward principles for the design of ontologies used for knowledge sharing", *International Journal of Human-Computer Studies*, v. 43, n. 5, p. 907-928, 1995.

[40] T. Mitchell, "*Machine Learning*", McGraw-Hill, Book, USA, 1997.

[41] X. Wu, J. Sun, "Study on a KQML-based intelligent multi-agent system" In: *Intelligent Computation Technology and Automation (ICICTA)*, 2010 International Conference on. IEEE, 2010. p. 466-469.