Chapter 3: Agent Architectures

Agent architectures can be classified according to various criteria (see section 2.2.4). For the purpose of this thesis, agent architectures are classified based on the reasoning model. In this chapter, an overview and a critique of each of the main classes of agent architectures are presented and discussed. A few definitions of agent architecture are given in section 3.1. Section 3.2 presents the historically first agent architecture: the symbolic reasoning agent architecture. Sections 3.3 and 3.4 discuss sub-symbolic agent architectures and hybrid agent architectures, respectively. Each of the presented architectures is initially discussed and criticised in its general form and then an example of such an architecture is described in greater detail. Section 3.5 proposes a hybrid agent architecture that is used in the INDABA agent architecture. Section 3.6 concludes this chapter with a comparison between the various presented models.

3.1 Introduction

The term agent architecture intuitively suggests a framework for the implementation of an agent. Agent architecture considers the issues surrounding the development and implementation of an agent based on a selected theoretical foundation. An agent architecture can be more formally defined as:

"[A] Particular methodology for building [agents]. It specifies how … the agent can be decomposed into construction of a set of component modules and how these modules should be made to interact…. An architecture encompasses techniques and algorithms that support this methodology" [106].

An alternative view on agent architecture is given as:

"[A] Specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more

abstract view of an architecture is as a general methodology for designing particular modular decomposition for particular tasks" [92].

There are various taxonomies proposed for agent and MAS architectures. The reader is referred to [38][90][57] for more details. In this thesis, agent architectures are classified according to the reasoning model used by agents.

In the remainder of this chapter, different agent architectures are presented and discussed.

3.2 Symbolic Reasoning Agent Architecture

Symbolic reasoning techniques are at the core of symbolic reasoning agent architectures. Section 3.2.1 presents a historical overview of the evolution of symbolic reasoning agent architectures, while section 3.3.2 presents some of the general characteristics and shortcomings of the symbolic reasoning approach.

As the representative of the symbolic reasoning agent architecture, one of the first implemented robots, namely Shakey [136], is discussed in section 3.2.3.

3.2.1 Introduction and History

Historically, the first agent architecture to appear, the symbolic reasoning agent architecture [136], has its roots in traditional artificial intelligence systems. An example implementation of a symbolic architecture is the early theorem-prover, General Problem Solver (GPS) [134]. Symbolic reasoning architectures are sometimes referred to as traditional architectures [87]. Expert systems are based on the symbolic reasoning paradigm. Successes of some of the symbolic reasoning systems, such as early expert systems (e.g. MYCIN [171]), have given credibility to the belief that such a paradigm can be easily extended to agent systems and embodied agents (robots). One of the seminal symbolic planning systems was STRIPS [65]. STRIPS is applied to agents and even to embodied agents, such as the robot Shakey [136]. Although the symbolic reasoning approach has been heavily criticised in the

late ‘80s (as discussed in section 3.2.2), the criticism did not stop the development of purely cognitive architectures such as SOAR [135] and ACT-R [4]. SOAR [135] and ACT-R [4] are both based on symbolic inference mechanisms. Agents systems that have utilised a symbolic planner as their main component include Integrated Planning, Execution and Monitoring (IPEM) [3] and “softbots” [62].

In the robotics field, after the initial success of Shakey [136] (which performed the required tasks, albeit slowly) and the harsh critique of symbolic reasoning systems during the ‘80s, there was not much development of pure symbolic reasoning, single agent systems. Similarly, the same applies to multi-robot symbolic systems. However, there were some simulated robotic systems based on a symbolic architecture, for example HOMER [193]. HOMER’s interaction with users was through commands that were given in a subset of the English language. Once commands were interpreted, the simulated robot would plan and execute given commands in its simulated environment.

It is becoming evident that any long-term artificial intelligence program must re-integrate some of the traditional AI based symbolic reasoning mechanisms [74]. The current trend is to create hybrid agent architectures where the symbolic component plays a significant role in agent architecture. An example of such a hybrid agent, 3T [22], is discussed in section 3.4.3.

The general characteristics of the symbolic reasoning agent architecture are presented in the next section.

3.2.2 General Characteristics of Symbolic Reasoning Agent Architectures

Symbolic reasoning agent architectures (also known as rational or deliberative) agents are based on a symbolic, abstract representation of the world and have an explicit knowledge base, often encompassing beliefs and goals [151]. Goal-oriented intelligent behaviour is explicitly coded into the agents, usually in production rule form. Typically, an agent can exploit many different plans to achieve its allotted goals. A plan is chosen on the basis of the current beliefs and goals of the agent. The

selected plan can be dynamically modified if these beliefs and/or the goals change. Rational agents can be considered advanced theorem-provers that manipulate symbols in order to prove some properties of the world. Implementing an agent as a theorem-prover allows the re-use of well-known techniques developed in the AI field, for example, the inference engines of expert systems.

The first obstacle that any symbolic reasoning architecture needs to overcome is that of an accurate implementation of the world model. The task of translating real-world entities and the often complex relationships between those entities into adequate symbolic representations is by no means a trivial task. Furthermore, there is no universal widely-accepted model for encoding the symbolic knowledge of the real-world. There are numerous methods in use, ranging from first order logic and production rules [87] to network representations, for example semantic nets [101].

The next problem that needs to be solved is which external stimuli, as sensed from the environment, can be ignored. Even for real embodied agents in real-world environments, information received via sensors is just a subset of all possible stimuli. For example, a robot may have a collision detector, but not a light sensor. The question that arises is whether a deliberative process will be different if a robot has more information about its environment. In other words, the choice of sensors that will influence the deliberative process is not always intuitive, and requires further research.

In real-world environments, symbolic reasoning suffers from the "real-time processing problem". To illustrate, consider that for real-world problems an optimal or nearly optimal solution is found based only on observation of the environment at initial time t. Most symbolic reasoning algorithms use a heuristic search of the problem space. However, because of a usually huge search space associated with real-world problems, an action executes during time t to $t+k$, where k is the time spent on finding an optimal or nearly optimal solution. In the meantime, during time k, the environment can change so that the optimal solution at time t may no longer be the optimal solution at time $t+k$.

Symbolic agent systems invariably have a lack of robustness to noise and inaccurate information [94]. In other words, symbolic agent systems do not degrade gracefully. This problem is common to most of the systems based on symbolic knowledge representation, for example, expert systems. As a result, symbolic systems usually perform well in simulated environments, but when implemented in a real-world environment, symbolic systems often fail to perform to their specifications.

One of the limitations of symbolic agents is that they execute sequentially. The sequential nature of a symbolic agent occurs due to the (essentially) sequential nature of the planning system that is at the core of symbolic reasoning architectures. Sequential execution of tasks may be acceptable for single agent systems, but in a MAS it is a serious shortcoming, due to the parallelism of MAS not being fully utilised.

Due to the shortcomings of pure symbolic reasoning approaches it is somewhat unlikely that pure symbolic reasoning architectures, on their own, will be predominant architectures of the future. However, symbolic reasoning techniques are widely used in the currently predominant architecture model, namely hybrid architectures (section 3.4).

3.2.3 Symbolic Reasoning Agent – Shakey the Robot

One of the first attempts at building a robot was a collection of hardware and software that was known as “Shakey the robot”[137]. Shakey used a symbolic planner, STRIPS [65], at its core. The next few sections discuss Shakey as an example of a purely symbolic reasoning agent.

3.2.3.1 Shakey – an Overview

Shakey was developed in the early ‘70s at Stanford Research Institute. The objective of the project was to incorporate vision, planning and the ability to learn into a single mobile robot. The main tasks that Shakey was designed to execute was to navigate from room to room and to push boxes, while avoiding obstacles.

Commands were given as "action routines" that operated at a very high level of abstraction. For example, the command Go_Thru (D1, R2, R1) meant "go from room R1 to room R2 via doorway D1".

Shakey used three sets of sensors:

- Bump detectors, implemented as touch sensors. The bump detectors were designed as antennas so that touch could be detected before the body of the robot touched an obstacle.
- An optical range finder, to provide Shakey with the distance from an object.
- A television camera together with image recognition software capable of recognising simple objects.

In addition to these sensors, Shakey also had a radio/video link to a stationary, off-board computer where the majority of processing was done.

Shakey was large by today's standards, having the size of an average-sized refrigerator and yet it had very little onboard intelligence [137]. Almost all processing was done on a mainframe using a radio link as a communication channel.

3.2.3.2 Shakey's Architecture

Intuitively, Shakey's architecture was well designed as it separated the actions performed into three groups based on "urgency" of the actions. Three action levels were implemented in Shakey's architecture, as illustrated in figure 3:

- Fast low-level actions (LLAs) that are represented by black lines on the figure 3. There are two types of LLAs. First type of LLAs are triggered by inputs from sensors without deliberation, similar to reflexes. Second type of LLAs are retrieval of rules and world model representation into a planner.
- Intermediate-level actions (ILAs), represented by broken grey arrows. ILAs represent simple, symbolic knowledge based actions.

- High-level actions (HLAs), represented by grey arrows. HLAs represent complex symbolic knowledge based reasoning, such as plans.

The soundness of Shakey's architecture was confirmed by the fact that Shakey could execute all of the envisaged tasks. Shakey could perceive its environment, plan and "reason" about its actions, and communicate. However, all of these tasks were executed excruciatingly slowly [35].

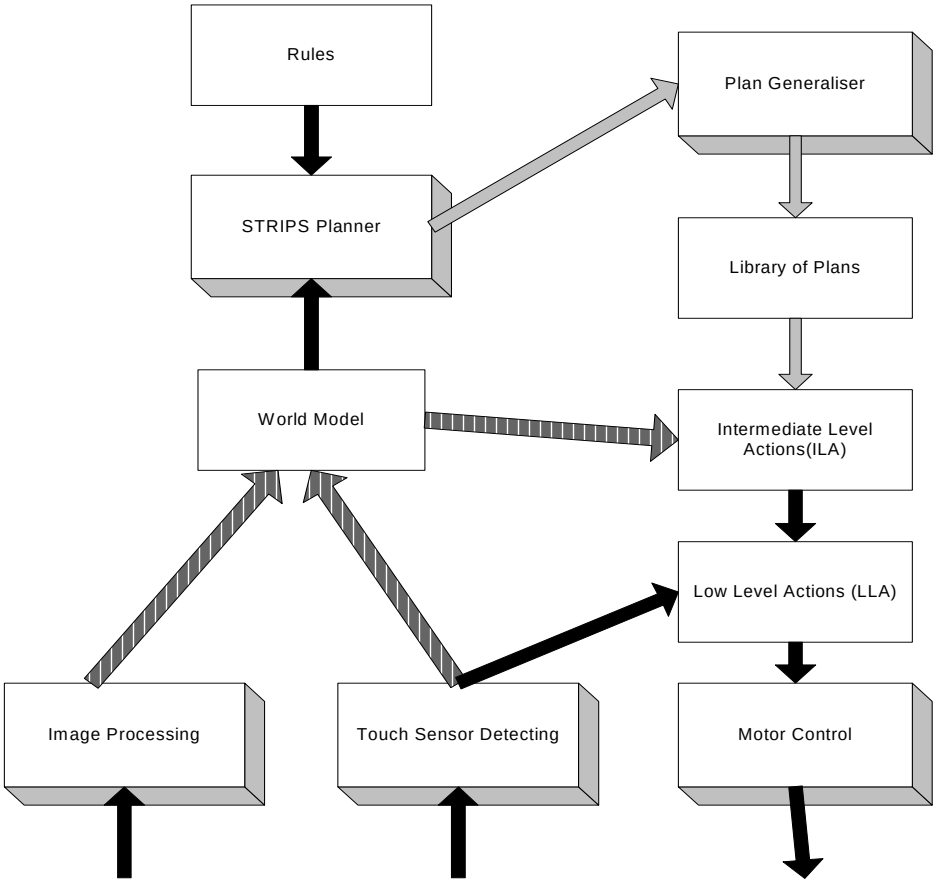

Figure 3. Shakey's Architecture, based on the description in [137]

The architecture is based on symbolic reasoning. All ILA and HLA actions are based on symbolic knowledge. However, it is interesting to note that the architecture included a sub-symbolic component as well. The creators of Shakey acknowledged a need for fast action-reaction couplings that are considered as reflexes. These "reflexes" are implemented as an LLA. An example of an LLA is that if touch sensors detect the proximity of an object, the touch sensor instantly sends a message to actuators (motors) to stop, without going through a symbolic reasoning deliberation process.

Although LLAs are executed without any deliberation, the symbolic representation of an actual situation is still created. The reason for creating a symbolic representation is to maintain a complete symbolic world model. For this purpose, the symbolic representation is sent to the upper levels that could deliberate upon the updated model.

Nilsson, one of the researchers that was involved in the Shakey project, refers to Shakey's architecture as a three-level architecture. Although the division of tasks is very similar to three-layer hybrid architectures that are described later in this chapter (section 3.4), Shakey's architecture should not be confused with hybrid three-layer architectures. In hybrid three-layer architectures there is a strong element of sub-symbolic based reasoning at lower layers, while Shakey's architecture is very much symbolic-oriented.

3.2.3.3 Shakey – Conclusion

Despite its limitations, such as slowness of its execution cycle, Shakey was a success. It performed the required tasks. More importantly, it gave significant insights on future development of robotic architectures.

The Shakey project has proven that:

"You could not, for example, take a graph-searching algorithm from a chess program and a hand-printed character-recognizing algorithm from a vision program and having attached them together, expect the robot to understand the world" [120].

Shakey has created a deeper understanding of problems associated with mobile, embodied agents. Shakey's approach to perceiving the world through its vision recognition software that could distinguish between boxes, walls and doorways was very advanced, especially considering the time period when Shakey was created.

One of the most important lessons learnt was that the creation of comprehensive world models was prohibitively computationally expensive. The majority of

computation was consumed in the transformation of sensor inputs to symbolic representations.

Other symbolic reasoning based robots were created, such as CART [127] and Hilare [78], but they all suffered from similar shortcomings, despite very simplified environments and the use of "state-of-the-art" symbolic reasoning mechanisms.

3.3 Reactive Agent Architecture

Not many papers have created such a reaction as the series of articles by Brooks [28][35][32] published in the early '90s. In these articles, Brooks delivered a harsh critique of the traditional AI approach to robotics. Brooks did not just criticise the traditional approach but also proposed, implemented and tested an alternative approach that has since become known as the subsumption architecture.

The subsumption architecture and its derivatives are often referred to as reactive architectures [130], a terminology adopted for the purpose of this thesis. Section 3.3.1 presents a historical overview of the evolution of reactive agent architectures, while section 3.3.2 presents some general characteristics of this approach. As the representative of reactive agent architectures, the original subsumption architecture [31] is discussed in section 3.3.3. A discussion of reactive agent architectures is given in section 3.3.4.

3.3.1 Introduction and History

Although a radical departure from then mainstream AI techniques, the subsumption architecture can be traced to another experiment from the mid-'80s. Braitenberg [26], as an experiment in cognitive science, proposed the development of 14 simple vehicles of varying characteristics. The first six vehicles had a very simplistic coupling of sensors to actuators and in a sense were very similar to the simple layers of the subsumption architecture. The subsumption architecture was proposed in 1986 [31] and it has been the inspiration for many other attempts and implementations [33][44][159][113]. Some implementations improved on the subsumption architecture

[44][159], while others used the subsumption architecture as the foundation that has led either to behaviour based robotics [113] or to hybrid systems such as [6].

Despite the fact that well-founded criticism had been levied against reactive agent architectures relatively soon after their appearances [84][95], further research in reactive architectures did not stop. Example applications of reactive architectures are that of Altenburg [2] (which is of special interest to this thesis as it is based on the same robotic platform as used in chapter 9), and Cog [33], again a project by Brooks and his team.

Although Cog initially exhibited some sophisticated behaviour, achieved through a basic reactive architecture, the Cog architecture had to incorporate some learning mechanism (implemented using neural networks) in order to achieve coherent behaviour [148].

Today, pure reactive architectures are not used in isolation due to the shortcomings that are presented in section 3.3.4. Reactive architectures have been superseded by behaviour based architectures, such as [113][68], not only in single agent systems, but also in hybrid systems [22][129].

3.3.2 General Characteristics

3.3.2.1 Origins of Reactive Architectures

Notwithstanding the similarity between Braitenberg vehicles [26] and their cognitive science approach, the major contributing origins of purely reactive architectures can be traced to two distinctive sources: biological sciences and engineering sciences.

From the biological sciences, the inspiration was drawn from the fact that the traditional notion of intelligence (i.e. the cognitive notion of intelligence) in biological systems has appeared very recently in evolutionary terms. The emergence of intelligent and cognitive thinking was preceded by millions of years of improving on

the interaction of biological systems with their environment. This gradual approach, which eventually results in emergent intelligent behaviour through interaction with environment and agents, was an inspiration to Brooks. Brooks focussed his research on the development of environment interaction mechanisms. Brooks states that:

"…mobility, acute vision and the ability to carry out survival-related tasks in a dynamic environment provide a necessary basis for the development of true intelligence" [28].

This view has been shared by other researchers [126][112]. From a biological perspective, the objective behind reactive architectures is then to create complete creatures that can exist in a dynamic people-populated world [34].

As such, creatures are dependent on efficient interaction with their environment. On the other hand, from an engineering point of view, it is imperative to develop efficient coupling between sensors and actuators, and the methodology that facilitates the development of such couplings.

The approach adopted for such couplings was uncompromisingly designed for speed and robustness. Unfortunately, the adopted approach was not sufficiently flexible and it has become the main reason for the limitations of reactive architectures, as discussed in section 3.3.3.3.

3.3.2.2 Underlying Concepts

Reactive architectures are based on the following fundamental underlying concepts:

- Situatedness: An agent is situated in its environment and directly interacts with the environment, without building a world model. This elegantly solves the problem of accurate world modelling and symbol grounding (as discussed in section 3.2.2). The agent is using the world as its model. In extreme interpretations of this architecture, the world is used even as a communication channel between the different layers of a reactive agent.

- Embodiment: Brooks argues that the only way to make sure that an agent can function in the real-world is if it has a physical body [28][35]. This view is shared for the purpose of this thesis.

- Intelligence: Reactive architectures adopt a bottom-up approach for intelligence modelling. Bottom-up approach means that basic layers are created first and then combined into more complex layers. Reasoning is also deemed unnecessary as it relies on a symbolic world model.

- Emergence: The main objective of a reactive agent architecture is that, through the agent's interaction with its environment, intelligent behaviour will emerge. It is very important to note that relatively simple agents that are not "aware" of their intelligence (they do not maintain any reasoning mechanisms) can exhibit emergent intelligent behaviour.

Brooks derives four ideas from each of the above concepts as an inspiration for the subsumption architecture [35]:

- "The world is its best model" – inspired by situatedness;
- "The world grounds regress" – inspired by embodiment;
- "Intelligence is determined by the dynamics of interaction with the world" – inspired by intelligence; and
- "Intelligence is in the eye of the observer".

The subsumption architecture is implemented as a set of layers that define agent behaviour. These layers are described next.

3.3.2.3 Layering in Reactive Architectures

The reactive architecture proposes building of simple layers based on augmented finite state machines. These layers are implemented as couplings between sensors and actions. All the layers execute in parallel and all the layers interact with the

environment. In other words, the reactive architecture is a horizontally-layered architecture.

The layering concept is demonstrated in figure 4. Three complex layers (the "collect" layer, "avoid obstacle" and "safe forward" layers) are implemented in the example. Only the simplest layer (the "safe forward" layer) is presented in detail. Layers that collect objects and avoid obstacles are abstracted.

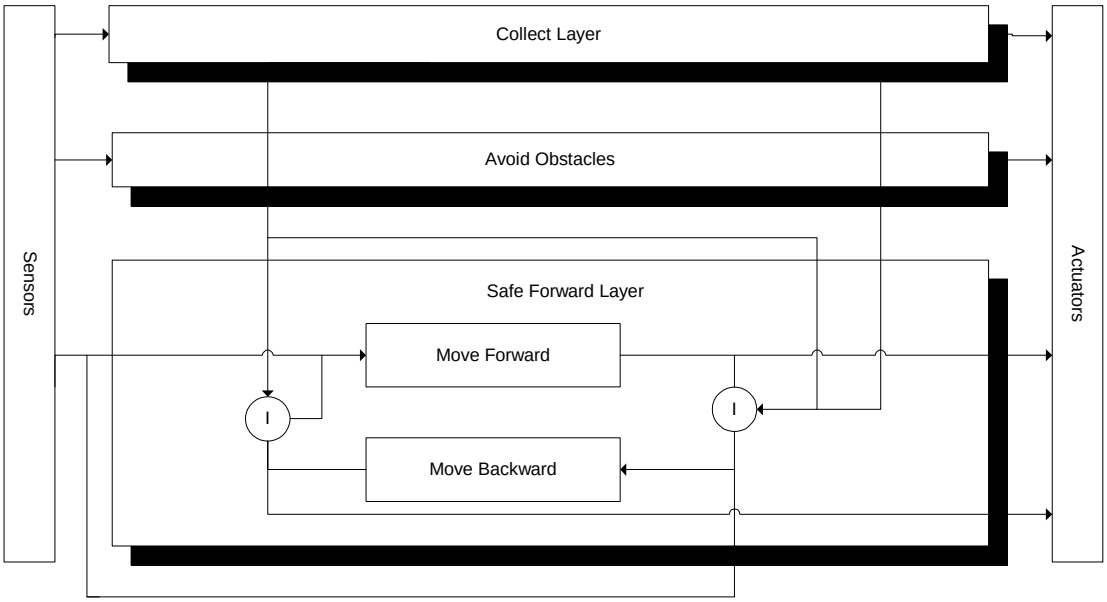

Figure 4. An example of the layers of a reactive agent

To illustrate the layering in reactive architectures, consider the simple actions implemented as layers "move forward" and "move backward". These two simple layers are directly coupled to actuators and sensors (see figure 4). In the reactive architecture, more complex actions are achieved through manipulation (inhibiting and enabling) of inputs and outputs of the appropriate layers. Examples of such layers are "avoid obstacle" and "collect" layers. The "avoid obstacle" and "collect" layers can enable and disable simpler layers such as "move forward" and "move backward".

Brooks advocates that the layering of a task-specific control can be achieved through this mechanism but it is unclear how this can be achieved without the decision on behaviours made at the time of design.

The inhibit and enable control mechanism is far from modular. More complex layers are tightly coupled with simpler layers. The consequence is that even a minor change at a simpler layer can have a severe consequence for the behaviour of a more complex layer and of the robot as a whole. Information-flow from the simpler layer to the more complex layer is non-existent.

The "safe forward" action is illustrated as follows: if the sensor input from the proximity detector is below the threshold (which is an indication that there are no obstacles), *move forward* is maintained and *move backward* inhibited. The moment the obstacle is detected, the situation gets reversed; *move forward* is inhibited and *move backward* initiated. The upper layer can influence lower layers as indicated in the figure.

3.3.2.4 Is a Reactive Agent Truly an Agent?

According to the definitions given in section 2.2.2, a straight answer cannot be given to this question. A reactive agent conforms to definitions as given in [197] [139], but not according to the definitions given in [82][80]. More importantly, considering the characteristics of agency as given in section 2.2.3, the answer is no. The first three characteristics of agency (autonomy, interaction and collaboration) can be (arguably) satisfied by a purely reactive agent, but the last characteristic, learning, cannot be satisfied.

In a purely reactive architecture there is no provision for any world model or internal state and therefore reactive architectures lack the basic fundamentals necessary for learning. Pure reactive agents do not learn.

3.3.3 Reactive Agent – Subsumption Architecture

Although it is arguably not a true agent architecture, the subsumption architecture is of such seminal importance, not only to the field of agent systems, but to the whole AI field, that it is discussed as an example of reactive architectures. Many robotic

systems were built based on the subsumption architecture and most of them were very successful [33][27][43].

Layers are implemented as finite state machines with four possible states:

- Output state. If a layer is in the output state, the layer outputs a message according to a computational transition and then switches to a predetermined state.
- Conditional Dispatch state. When a layer is in the conditional dispatch state, the layer tests the value of a function and then switches to one of the predetermined states.
- Self state. When in a self state, the layer performs a computation that affects the internal state (albeit limited to a few variable registers).
- Event Dispatch state. In the event dispatch state, a layer waits for event(s). Once an event occurs, the layer switches to a predetermined state.

Each layer has input and output that can be affected by suppressor and inhibitor connections respectively. If the inhibitor is active (a message going through the wire) then the outputs of the corresponding layer are suppressed. If the suppressor is active, then the input is disregarded. A typical black box representation of a layer is given in the figure 5.

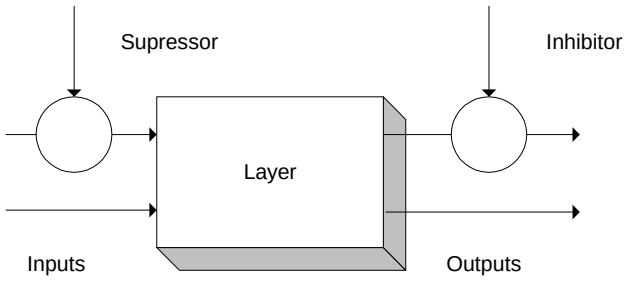

Figure 5. Black Box Approach for a Reactive Agent Layer

Suppressing inputs and inhibiting outputs of simpler behaviour achieves more complex layers. The whole methodology for building agents based on the subsumption architecture is based on the iterative approach as described next.

Simple layers are built and tested in the real-world environment. Once successfully tested, the simple layers serve as building blocks for more complex layers that are then tested in the real-world environment. Again, these layers now serve as building blocks for even more complex layers and the whole process is repeated until the desired behaviour is achieved.

A reactive architecture that are implemented according to the above described methodology conforms to the goals that are stipulated by [28]:

- The capabilities of the agent are built up in small incremental steps and at every step there is a complete system that can be tested.
- At each step, the embodied agent is tested in the real-world environment.

As indicated at the beginning of this section, there were some notable successes of this architecture. The reactive architecture has probably reached its pinnacle in the implementation of the robot, Herbert [29]. Herbert incorporated mobility, image recognition and robotic arm coordination. Its tasks were to wander around offices and to collect soda cans. According to Gat [74], Herbert has never reliably performed the desired task.

3.3.4 Subsumption Architecture - Conclusion

The subsumption architecture deviated from the traditional AI (symbolic reasoning) systems. The traditional AI systems usually implemented the symbolic reasoning mechanism, the Sense – Think - Act cycle [148].

There were many problems with the symbolic reasoning approach as discussed in section 3.2.2. The core of the problem lies in the drastic simplification of the symbolic model of the world, which is necessary to reduce the search space of the planner. Reactive architectures have eliminated this problem. Reactive architectures do not maintain a world model; instead, reactive architectures indirectly use the world as its own model.

Although the reactive architecture approach is good for simple agents and behaviours, it is, unfortunately, inadequate for more complex tasks mainly due to the lack of any world model. The lack of world model in the subsumption architecture has introduced the following problems:

- Agent reasoning is based purely on sensor readings from the local environment. This can lead to a local optimum solution (a course of action) that is not the global optimum solution (the best possible action).
- It is difficult to see any possibility for learning from experience or from other agents.
- The idea of emergence is valid, but reactive architectures do not provide mechanisms for recognition and incorporation of such emerging behaviour. Instead, all layers are handcrafted, so any new layer (behaviour) would require reprogramming.
- Interdependencies between the layers can become unmanageable in case of many layers, due to the numerous couplings between layers.
- With MASs in mind, the lack of direct communication between the agents (even between the lower layer to the upper layer) can lead to negative interaction, i.e. conflict. The lack of communication (except through the world itself) prevents implementation of any coordination mechanism.

Although the shortcomings of the subsumption architecture are numerous, the subsumption architecture, and reactive architectures in general, played a major influential role in today's embedded agent predominant architectures, namely the hybrid architectures. Due to the reactive architectures' superb interaction capabilities with the real-world, reactive architectures (or their derivatives) are often used as the simplest layer of hybrid architectures.

Reactive architectures should also be considered in relation to the time period when they appeared. The computational cost in terms of hardware has dropped tremendously in the last 20 years and computational power has increased almost exponentially. This development can influence the validity of the "do not think (because it is costly and time consuming) – act!" premise of reactive architectures.

Many robots [33][27][43][29] have been built based on the subsumption architecture, some of them very successful. In a sense, these early successes have created a renewed interest in robotics.

The root of the shortcomings of the subsumption architecture can be found in the definition of the subsumption architecture, as given by Brooks. Brooks defined the subsumption architecture as a parallel and distributed computation formalism for connecting sensors to actuators in robots [34]. The strong engineering influence in the development of the subsumption architecture is evident from this definition. In other words, the subsumption architecture is mainly concerned about hardware efficiencies without much concern for higher concepts such as models, learning capabilities and sociality between agents.

It can be claimed, with confidence, that without the subsumption architecture and the pioneering work done by Brooks and his team, robotics would be far from the capabilities that are demonstrated today.

3.4 Hybrid Agents

As discussed in sections 3.2 and 3.3, purely reactive and symbolic agent architectures have different shortcomings and benefits. It was natural that further research led to the emergence of hybrid agents which attempt to exploit the strong points of each approach.

Purely reactive architectures have been criticised for their inability to perform complex tasks [88]. Reactive architecture-inspired approaches, such as behaviour based robotics [81][109], have replaced purely reactive architectures. Section 3.4.1 tracks some of the hybrid agent history, and section 3.4.2 overviews the typical three-layer architecture that has become almost standard for hybrid agent architectures. Section 3.4.3 describes an example of hybrid agent architecture in greater detail.

3.4.1 Introduction and History

Once the critique of purely non-symbolic architectures appeared [84][95], various attempts were made on improving on non-symbolic architectures [159][165]. By the early nineties at least three teams of researchers [73][45][23] had independently proposed true hybrid architectures, consisting of three layers.

These early hybrid agent architectures have evolved over time, being continually improved. For example, Bonasso's work has evolved into the three-tier architecture, 3T [22] and Gat's has evolved into an architecture called ATLANTIS [73]. Both of these architectures have been successfully applied in robotic systems. 3T has been used as the core architecture for NASA Johnson Space Center's Robotic Architecture robot that was able to recognise people [22]. ATLANTIS was also implemented in a number of robotic systems [73][71].

InterRAP [129], another important hybrid agent architecture, combines not only reactive and symbolic planning aspects of agents but also the social aspects. It is also interesting to note that ATLANTIS and 3T have their origin in robotics, while InterRAP has its origins in DAI [90].

The more recently proposed hybrid architectures is Jet Propulsion Laboratory's (JPL) CLARAty [194]. The importance of CLARAty is that it proposes the use of existing methodologies, software and approaches such as open source software libraries and object-oriented design methodologies (e.g. the Unified Modelling Language (UML)).

In addition to the aforementioned architectures, there are also various design tools that support hybrid architectures. Two of the design tools that support hybrid architectures are mentioned here for the sake of completeness. DESIRE [58] is a methodology that provides basic modularisation techniques that can be used for building hybrid agent systems. DESIRE provides for horizontal and vertical layering approaches (as described in the next section). Task Description Language (TDL) is a software development tool [173] that is implemented in the C programming language as an extension, and can be used for the development of three-layer architectures.

3.4.2 General Characteristics of Hybrid Agent Architectures

In this section, some of the general characteristics of hybrid agent architectures are overviewed and briefly discussed.

3.4.2.1 Layered Architectures

Most hybrid systems are implemented using layered architectures. Architectures can be layered horizontally or vertically. In the horizontal layering approach all layers are at the same level and execute independently. Vertical layering means that there is a number of layers between the sensors and actuators, while in horizontal layering there is only one layer that has as inputs sensor inputs and as outputs actions. Brook's original subsumption architecture follows a horizontal layering approach [31]. An illustration of a horizontally-layered architecture is given in figure 6.

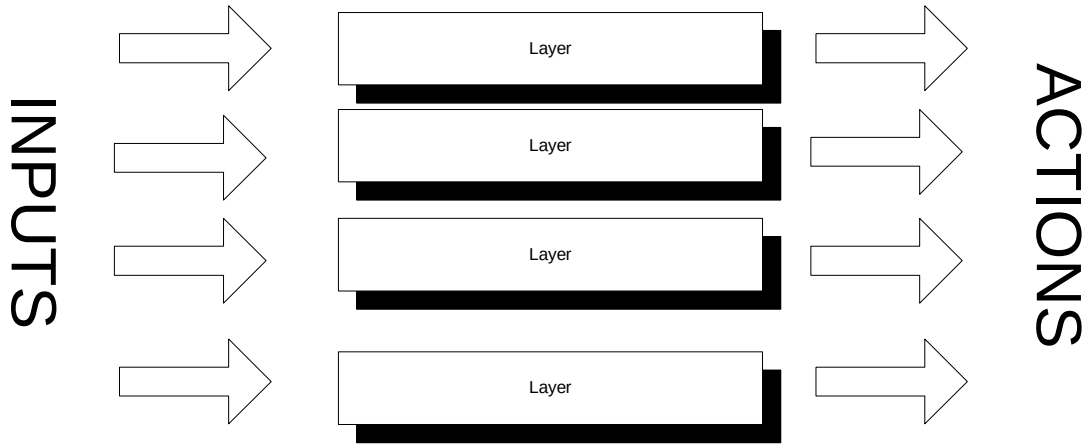

Figure 6. Horizontal Layering Agent Architecture

In the vertical layering approach layers are hierarchically ordered with the complexity of layers increasing with their level. Interaction between layers is defined as hierarchical. In other words, interaction between the bottom layer and the top layer cannot be direct. The interaction has to be done through intermediate layer(s), whereas in the case with horizontal layering, all layers execute in parallel. An illustration of a vertically layered architecture is given in figure 7.

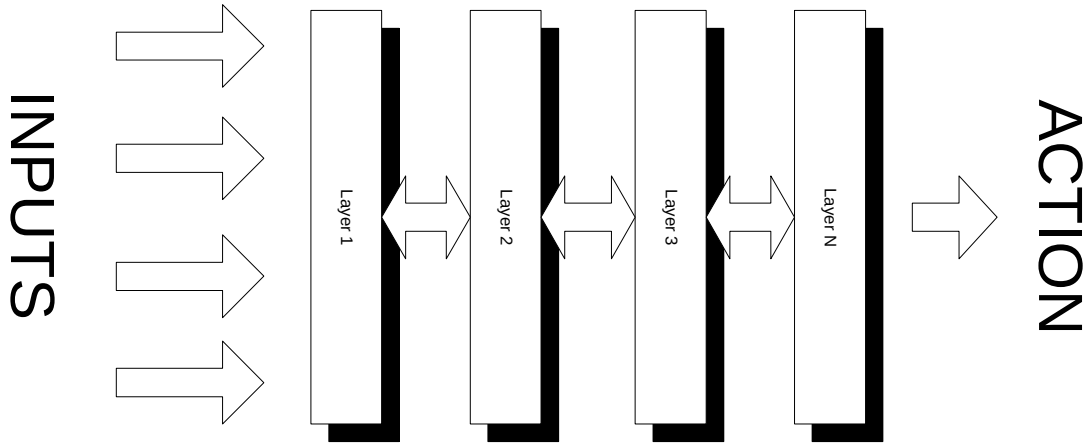

Figure 7. Vertical Layering Agent Architecture

Section 3.4.1 gave frequent reference to three-layered architectures. It seems, especially in the autonomous embodied agents field (robotics), that most researchers [22][73][173] have standardised on the use of three vertical layers. The reason for this is not so much theoretical, but based on a pragmatic approach and on the results of experimenting with various numbers of layers. Most hybrid agent architectures consist of a deliberative layer, a reactive layer and an interface between them. It is noted that, although the CLARAty architecture is a two-layer architecture, it is logically very similar to a three-layer architecture. An example of a typical three-layered architecture is given in figure 8 (modified from [22]).

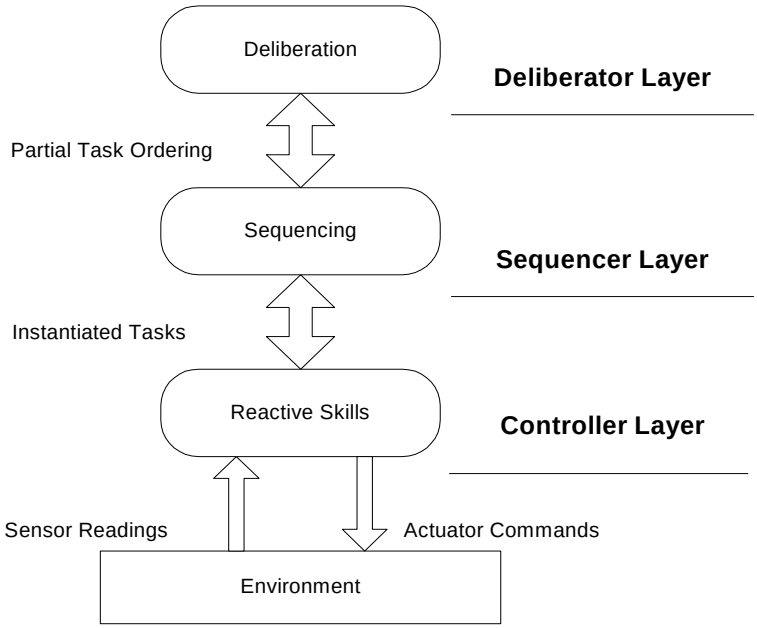

Figure 8. A Typical Three-Layer Agent Architecture

There are numerous, sometimes confusing naming conventions of these three layers. For the purpose of this thesis the adopted terminology is that of the ATLANTIS architecture [74]. The layers in the ATLANTIS architecture are called the controller, the sequencer and the deliberator layers. Each of these logical layers is described in greater detail in the next few sections. Naming conventions for the different architectures are summarised in Table 1.

Architecture	Top Layer	Middle Layer	Bottom Layer
ATLANTIS	Deliberator	Sequencer	Controller
3T	Planning Layer	Sequencing Layer	Skill Layer
TDL	Planning Layer	Executive Layer	Behaviour Layer

Table 1. Overview of Three-Layer Architecture Terminology

3.4.2.2 Controller Layer

The main purpose of the controller layer is to react dynamically, in real time, to changes in the environment. It can be seen as the implementation of fast feedback control loops, tightly coupling sensors to actuators [74]. The controller layer is usually implemented using a behaviour based robotics approach [109], as a set of behaviours. Behaviours (in behaviour based robotics terms) can also be seen as control laws that encapsulate sets of constraints in order to achieve specific behaviour [113].

Behaviours are usually implemented as handcrafted, simple, conditional rules. The behaviours take, as conditions, sensor readings and as action the behaviours provide motor actuation. The need to handcraft the behaviours is one of the challenges of behaviour based robotics that needs to be overcome. Because the behaviours interact with the real-world environment in real time, it is of crucial importance that these behaviours are fault-tolerant (or at least fault-aware) and that behaviours are bound by a maximum deliberation time. Behaviours are usually stateless; they do not maintain any local or global environmental models. Behaviours at this level are basic (primitive) behaviours. Basic behaviours are often combined into more complex behaviours by the next layer, the sequencer layer. Basic behaviours can be selected

either based on researchers' experience or according to a methodology such as that described in [113].

3.4.2.3 Sequencer Layer

The job of the sequencer layer is to manipulate basic behaviours into more complex behaviours that are closer to the symbolic layer, i.e. the deliberator. The sequencer layer achieves this task by enabling or disabling behaviours and/or by providing parameters for behaviours' execution. The storage mechanism for more complex behaviours is usually a library of plans for implementation that use basic behaviours. The task of breaking down a complex behaviour into basic behaviours is by no means trivial.

There are two main approaches to transform complex behaviours into basic behaviours:

- The universal plan approach, where all of the states and hard-coded complex behaviours are enumerated as a combination of basic behaviours, usually in table format [165].

- The conditional sequencing approach, where only the conditions that trigger behaviours are stored. Conditional sequencing can be implemented using either special, purpose-designed languages such as RAP [68] or as an extension of more traditional programming languages such as C, as was the case with TDL [173].

The sequencer layer can be seen as the interface between symbolic knowledge representation (as implemented in a deliberative layer) and sub-symbolic knowledge representation (as implemented in a controller layer).

3.4.2.4 Deliberator Layer

Knowledge representation in the deliberator layer is symbolic in nature. All reasoning is done using a symbolic world model. The symbolic reasoning approach and the

problems associated with this approach were discussed in section 3.2. The deliberator layer is the most abstract layer as it does not have direct interaction with the environment. However, the deliberator layer performs some of the crucial tasks in a hybrid agent architecture. The task of the deliberative layer is to perform the following functions:

- to build and maintains the world model,
- to deliberate (reason) on the course of action in symbolic terms, and
- to interface with the sequencer layer.

Being based on symbolic knowledge representation for its deliberation, the deliberator layer usually uses traditional artificial intelligence techniques such as planning and inference [197][72]. These techniques are traditionally computationally demanding and thus the deliberative layer does not respond to changes in the environment in real time. The controller, and to a lesser extent the sequencer layer, operate in real time. The deliberator expresses medium to long-term goals to the sequencer layer.

The deliberator layer interfaces with the sequencer layer either through plans that are presented to the sequencer layer, or responds to queries from the sequencer layer [74].

The deliberator layer is usually implemented in a standard, high level programming language, or using an inference engine (e.g. an expert system shell).

3.4.3 Hybrid Agent Architecture – 3T

In this section, an example of a hybrid agent architecture is presented and discussed in greater detail. The name 3T is derived from the three layers used by the architecture. An introduction to the 3T architecture is given in section 3.4.3.1, followed by an overview of the layers of the 3T architecture in sections 3.4.3.2 to 3.4.3.4. Section 3.4.3.5 provides a summary of the 3T architecture.

3.4.3.1 Introduction and an Overview

Most of the team members that created the 3T architecture [22] were involved with NASA and some of the 3T applications were related to space research programmes [22].

The 3T architecture was built upon various earlier research efforts [67][73] done by the same team, and designed with applications in embodied agents (robots) in mind. The three layers of the 3T architecture correspond to the layers that are described in sections 3.4.2.2 – 3.4.2.4.

For the purpose of the 3T overview, the original 3T terminology is used when describing the layers.

3.4.3.2 Skills Layer

The task of the skills layer, being the bottom layer of the 3T architecture, is to interact with the environment. A skill corresponds to a behaviour and its purpose is to achieve or maintain a particular state. Because the skills are dependant on the robot's physical implementation, any hard coding would seriously limit the architecture's flexibility. The approach was thus taken to implement a robot-independent skill representation based on work by Yu *et al* [202]. The representation consists of:

- Inputs and outputs, that are declarative descriptions of expected inputs and produced outputs. Outputs of one skill can be linked to inputs of another skill, thus allowing for chaining of skills.
- A computational transform, which forms the core of a skill, and which produces outputs according to computational rules, from inputs.
- An initialisation routine, that allows for skill initialisation in a secure and expected manner.
- An enable/disable function, that provides a sequencer with a mechanism to suppress and to enable a skill.

The enabled skills are executed in parallel. All skills interface with a skill manager that in turn interfaces with a sequencer, providing the sequencer with a single entry point to the skill layer. The skill manager handles all communications, asynchronous events handling and the enabling/disabling of skills.

3.4.3.3 Sequencing

The sequencing layer in 3T is implemented as a Reactive Action Packages (RAP) [67] interpreter. RAP is a LISP-based structure that is simply a description of the desired task to be achieved. It is important to note that a task is not unconditionally described in minute detail. The task description relies on the robot's perception of its environment. In other words, depending on the environmental perception (model), the task might be executed in a different manner. Each RAP has a sequence of skills that is either activated or deactivated in order to achieve an allocated task. This mechanism provides a sequencer to the skills layer communication mechanism. Specialised skills, called events, provide a communication channel from skills to sequencer layers. Events provide a feedback by communicating the perceived state of the environment. The sequencer layer uses this information to determine if a particular set of skills have been completed.

3.4.3.4 Planning

The sequencer layer of the 3T architecture does not perform optimised resource allocation nor does it organise the sequence of routine tasks that perform more complex and more useful tasks. These are the tasks of the deliberator layer. The planning layer operates on a higher level of abstraction than the sequencing layer. There are a few reasons for this level of abstraction. Firstly, a higher level of abstraction is more understandable by humans and the planning layer can often serve as a system-user interface. Secondly, a higher level of abstraction reduces the size of the search space.

The Adversarial Planner (AP) [60] is used in 3T. The planning takes the form of higher level RAP that are then decomposed into more elementary RAPs by the sequencer layer.

AP has two features that are considered very useful in robotic applications: it has the ability to control more than one agent at the same time and it can reason with agents that exhibit negative interaction (adversary attitude), for example agents that are not controlled by 3T (uncontrolled agents in 3T terminology).

Since the focus of this thesis is an architecture for cooperation between agents, the first feature is covered in more details. The multi-agent coordination mechanism in 3T allows for coordination between robots, but its usefulness is questionable. All coordination must be done through a central system. In other words, instead of having agents cooperate through consensus or some other coordination mechanism, it imposes a centralised, hierarchical control on otherwise autonomous agents. This severely restricts the potential of novel, self-organising multi-agent applications. Therefore, this thesis treats the 3T architecture as mainly a single-agent architecture.

3.4.3.5 3T – Conclusion

3T is a comprehensive architecture with some interesting features, for example the deliberator layer adversarial planning and its (albeit limited, as discussed in the previous section) provision for the coordination of multiple agents. Coupling between layers is coherent from the top to the bottom layers, but the upward flow of information is very limited.

The sequencing and deliberator layers are implemented in LISP. Although LISP has been one of the most commonly used programming languages in AI research, its applicability to real-world embodied agents is questionable. LISP has relatively large resource requirements (it is usually implemented as an interpreter) and its speed of execution is usually slower than that of compiled programming languages. Portability to different hardware, e.g. lower-end platforms (such as a cheap, swarm like robotic

system [185][184]) might also present a problem, due to the relatively high computational demands of a LISP interpreter.

3.5 Summary

The focus of this chapter was the application of various agent architectures to robotic applications. This chapter overviewed three types of agent architectures, namely symbolic, reactive and hybrid. These architectures were initially discussed in general terms. The general discussion was followed by a detailed review of representative examples of the three architectures.

The next chapter naturally follows this chapter by extending the discussion and overview to MAS architectures.