Chapter 5: New INtelligent Distributed Agent Based Architecture

As seen in chapters 3 and 4, agent architectures are almost as diverse as agent applications. Currently there is no architecture that will suit all applications. The new proposed INtelligent Distributed Agent Based Architecture (INDABA) is designed with the goal of constructing an architecture for cooperative, embodied agents (robots) in multi-robot teams. The architecture presented in this chapter is mainly a conceptual framework that is not too prescriptive in implementation technique. Instead, INDABA should be seen as a guideline for designing cooperative agents. An overview of INDABA and rationale behind INDABA are given in section 5.1. Section 5.2 presents the first layer of INDABA, the controller layer, together with an example that illustrates a potential implementation. The second INDABA layer, the sequencer layer, is presented in section 5.3, again together with an example that illustrates a potential implementation. Section 5.4 presents the concept of deliberator layer with an example that illustrates its workings. The last INDABA layer, the interaction layer, is presented in section 5.5. The example used to illustrate the interaction layer and associated concepts is based on a cooperative problem-solving approach that consists of five steps, as described in section 5.5. Section 5.6 summarises INDABA and outlines possible future developments.

5.1 Overview of INDABA

Based on the investigations of agent architectures that were presented in chapters 3 and 4, the following was observed:

- The hybrid architectures have clear advantages over pure symbolic and reactive architectures as they have the best characteristics of both approaches and they address the weaknesses of both approaches.
- Due to the various agents' (and robots' in particular) technology platforms, as well as the numerous possible applications, it is not possible to have a unified, general-purpose architecture that will satisfy all requirements.

- The coordination mechanism in the majority of existing architectures is not flexible enough or virtually non-existent.
- The coordination mechanisms often ignore uncertainty about a task by assuming the ideal environment where details about the task are complete and accurate.

With these findings in mind, the new proposed conceptual architecture, INDABA, was designed and developed. INDABA is a layered architecture. It appears that most researchers, i.e. Brooks and Barnes [31][15] agree that agent architecture should be layered [106]. Furthermore, in the field of autonomous robots, it seems that most of the researchers [22][73][173] have standardised on hybrid architectures, consisting of three vertical layers.

As discussed in section 3.4.2.1, architectures can either be vertically or horizontally layered. INDABA provides for a hybrid between these two approaches, albeit more of a vertical layering approach than horizontal. The layers of INDABA are illustrated in figure 12. In INDABA, the main interaction between the layers is between the vertical layers, while less frequent interaction between the agents is done through the horizontal layers. The lack of interaction between layers was one of the main critiques of many horizontal-layering architectures, such as the subsumption [31] and behaviour based architectures [113].

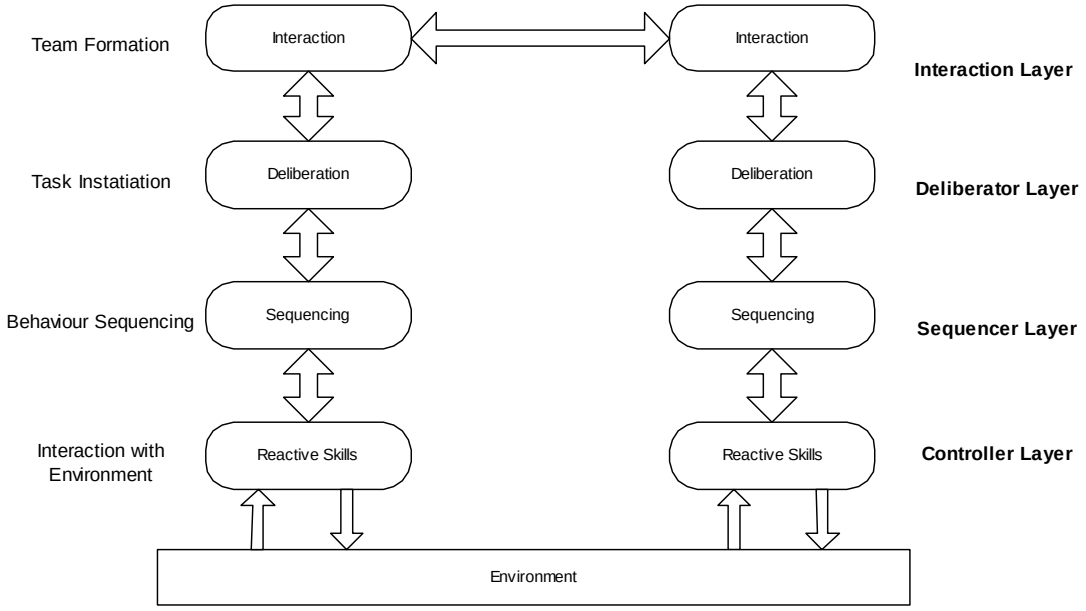

Figure 12. INDABA Layers

In comparison with more common three-layer architectures such as 3T [22] and ATLANTIS [73], INDABA introduces an additional layer; the interaction layer, that facilitates coordination through task allocation.

INDABA is designed with ease of coordination between the agents in mind. Ease of coordination between agents was achieved through introduction of a coordination-oriented layer that encapsulates the selected coordination mechanism.

INDABA is also a hybrid architecture. Currently, the most common approach to design robots is to use a hybrid approach (as described in 3.4) that combines the best characteristics of reactive and symbolic architectures (see sections 3.2 and 3.3 respectively). As indicated in table 1, different architectures use different layer naming conventions. INDABA adopts the layer names deliberator, sequencer, controller and interaction layers.

Each of the INDABA layers is discussed next, starting with the lowest level layer, i.e. the controller layer.

5.2 Controller Layer

The main purpose of the controller layer is to react dynamically, in real time, to changes in the environment. The controller layer can be seen as the implementation of fast feedback control loops, tightly coupling sensors to actuators [65]. In hybrid architectures [22][73], the controller layer is usually implemented as a set of behaviours using a behaviour based approach.

INDABA also implements the controller layer using behaviours. Behaviours implemented in the controller layer are basic (primitive) behaviours. Basic behaviours can be combined into more complex behaviours. Basic behaviours can be selected either according to the researcher's experience or according to a methodology such as described in [113].

Implementation of the behaviours in INDABA follows the guidelines given by Matarić [113]. Generation of simple behaviours such as *move_forward*, *turn_left*, *turn_right* and other simple behaviours, precedes the synthesis of more complex ones such as *avoid_obstacle*, *go_to* etc.

An example of a simple behaviour, *move_forward*, is given in algorithm 2 for a simple differential drive robotic platform, such as Lego Mindstorm [185].

```
behaviour move_forward
  while true
    LeftMotor(On)
    RightMotor(On)
  end while
end behaviour
```

Algorithm 2. move_forward behavior

Most of the simple behaviours have no limit on duration of their execution. As such, simple behaviours are controlled by more complex behaviours. More complex behaviours can start and stop simpler behaviours.

Complex behaviours usually have an associated completion condition. If the completion condition is satisfied, then the complex behaviour terminates. Alternatively, a complex behaviour might have a resource restraint that governs its execution. For example, a complex behaviour can be allowed to execute only for a limited period of time. In other words, behaviours can stop their own execution if stopping criteria are defined and the stopping criteria are met.

To illustrate a complex behaviour, consider an implementation of the *avoid_obstacle* behaviour. The *avoid_obstacle* behaviour in this example is activated if touch sensors detect an obstacle. The value for the wait function has been arbitrarily set to 50 (the parameter value indicates the hundredth part of a second, so a value of 50 indicates 0.5 seconds). The *avoid_obstacle* behaviour has a limited duration of its execution. The behaviour is implemented as a combination of simpler behaviours *move_forward,*

move_backward, turn_right, turn_left, detect_left_touch and *detect_right_touch*, as illustrated in algorithm 3.

```
behaviour avoid_obstacle
  if detect_left_touch
     start move_backward
     wait (50)
     stop move_backward
     start turn_right
     wait(50)
     stop turn_right
  else
     if detect_right_touch
       start move_backward
       wait (50)
       stop move_backward
       start turn_left
       wait(50)
       stop turn_left
   end else if
end behaviour
```

Algorithm 3. avoid_obstacle behaviour

It is important to note that INDABA does not prescribe specific implementations of behaviour. Implementations usually depend on the robotic platform, and INDABA's goal is to provide a platform independent architecture. Each behaviour is treated as a black box and as an autonomous, self-contained object. Using such approach, a multitude of various platforms, some of them with existing comprehensive libraries of behaviours, can be easily encapsulated into an INDABA agent architecture.

While higher layers can be interchangeable between various hardware and software platforms, the controller layer is platform-dependent, because it executes on a specific robot platform. The implementation of the controller depends on a physical suite of

sensors and actuators. Chapters 7, 8 and 9 discuss specific implementations of INDABA. The first two implementations runs as a set of algorithms in a simulated robot environment, and the third as a set of behaviours executing on a robotic platform.

5.3 Sequencer Layer

The job of the sequencer layer is to further combine behaviours into more complex behaviours that are closer to higher level goals. The sequencer layer achieves this task by enabling or disabling behaviours and/or by providing parameters for the execution of behaviours. The complex behaviours in the sequencer layer are seen as sub-tasks, used by a symbolic reasoning mechanism implemented in the next layer, the deliberator layer.

As discussed in section 3.4.2.3, there are different ways in which the sequencer layer can be implemented. For the initial implementation of INDABA, the universal plan approach [165] was adopted. The sequencer layer is based on a universal plan in the form of a table that is loaded from a text file. Each sub-task has a set of corresponding active behaviours and a condition or set of conditions that will satisfy its goal. The conditions are usually represented as a combination of completions of simpler behaviours.

Table 3 illustrates an implementation of a sequencer layer. Behaviours *safe_wander*, *detect*, *collect* and *home* are complex behaviours that are implemented in the controller layer, while sequencer layer behaviours ***Find***, ***Collect*** and ***Home*** are implemented as a combination of these complex behaviours in the controller layer (refer to table 3).

Sub-task	Active Behaviours				Goal
	safe_wander	*detect*	*collect*	*home*	
Find	1	1	0	0	*detect*
Collect	0	1	0	0	*collect*
Home	0	0	0	1	*home*

Table 3. Illustration of INDABA sequencer layer

For the example illustrated in table 2, it is important to note that the *detect*, *collect* and *home* behaviours do have completion conditions, while *safe_wander* does not.

The sequencer layer can be seen as a higher abstraction of basic behaviours. The result is a group of sub-tasks that can now be instantiated from the deliberator layer.

5.4 Deliberator Layer

The next layer of INDABA is the deliberator layer. The deliberator layer is the first INDABA layer that uses symbolic reasoning based on a symbolic world model. However, the deliberator layer performs crucial functions in INDABA (as for any other hybrid agent architectures):

- Builds and maintains the world model.
- Deliberates (reasons) on a course of action in symbolic terms.
- Interfaces with the sequencer layer, by starting s in the sequencer layer.

The initial INDABA implementation uses a simple backward chaining inference engine and a rule database to implement the deliberator layer.

To illustrate the working of the deliberator layer, consider a simple foraging problem. For this purpose, the sufficient set of rules are:

Task: **FORAGE**
Rule1: IF ***Find*** THEN ***Collect***
Rule2: IF ***Collect*** THEN ***Home***

The rules are loaded from a text file. The process start by pursuing the goal ***Home***. Simple backward chaining leads to goal ***Find*** .

Goal ***Find*** is then sent to the sequencer layer. The sequencer layer then performs a table look-up to determine the active behaviours associated with goal ***Find*** and the stopping criteria. From table 2, the active behaviours for goal ***Find*** are *safe_wander*

and *detect*. The stopping criterion for goal ***Find*** is that the behaviour *detect* is completed.

If the *detect* behaviour is completed, an object is detected and the sequencer layer reports to the deliberator layer that it has achieved its given goal, i.e. ***Find***. The deliberator layer then, from Rule 1, inferences that the ***Collect*** behaviour needs to be satisfied. The ***Collect*** goal is then passed to the sequencer layer, and the execution continues.

The above example is simple, but sufficient to describe basic execution of a task in INDABA. It is important to note that in the implementation of INDABA, as presented in this thesis, the deliberator layer does not build its own world model.

5.5 Interaction Layer

The interaction layer encapsulates mechanisms that facilitate the coordination between agents in INDABA. The interaction layer maintains its own internal state by means of maintaining multiple variables. The variables in INDABA are divided into sets that are referred to as mental states. In INDABA, each agent maintains its own set of mental states. There are three separate sub-sets of mental states, namely: self-related, task-related and society-related mental states.

Mental states can be changed by the agent itself, based on the agent's own experience, or they can be changed through interaction with other agents. Each of the mental states implemented in INDABA are described next.

5.5.1 Self-Related Mental State

The self-related sub-set of mental states consists of the agent's beliefs about its own capabilities. The initial application of INDABA is in robotics, where this sub-set consists of a robot's enumeration of its own sensors and actuators and their characteristics. In INDABA, the self-related mental state is used to determine the agent's own suitability to a task. For the purpose of this thesis, a simple hard-coded

data structure was used to represent the self-related mental state. An example of such data structure is discussed in chapter 7, where a particular INDABA implementation is presented in detail.

5.5.2 Task-Related Mental State

Once allocated a task, the agent must store the task information. All information related to a task is stored in the agent's task-related mental state. A task is described by a set of attributes. For the purpose of this thesis, task-related mental state was implemented as a simple data structure, as discussed in chapters 7, 8 and 9.

It is important to note that a simple data structure is not the ideal implementation. A simple hard coded data structure implies prior knowledge about the problem domain. INDABA is not a prescriptive framework, but allows implementations using more flexible mechanisms, such as KQML [66][99] and XML [186].

5.5.3 Society Related Mental State

The implementation of a society related mental state depends on the selected coordination mechanism. For example, in a pure auctioning coordination mechanism, the society mental state consists of only one parameter, namely the cost of an agent. On the other hand, other approaches such as the hierarchical approach (described later in section 6.3.2) and a social networks based approach (refer to section 6.6) require more complex data structures.

For the purpose of the experiments presented in this thesis, the society related mental state was implemented with a social networks based approach in mind. The implementation provides for the creation and maintenance of two distinctive types of social relationships between the agents (described in greater detail in chapter 6), through array-type data structures.

5.5.4 Coordination

Coordination in INDABA is achieved through a cooperative problem-solving process. Neches *et al.* [132], Wooldridge *et al* [198] and Genesereth *et al* [75] divided the cooperative problem-solving process into four main stages:

- **Potential Recognition**: where an agent investigates which agents are capable of executing the task and tasks are allocated.
- **Team Formation**: where agents start to share a common goal and self-organise to form teams to achieve this common goal.
- **Plan Formation**: where an agent or a whole team decides on the division of a goal into sub-tasks and on the allocation of those sub-tasks.
- **Plan Execution**: where an agent or, in the case of a team, agents execute their allocated sub-tasks.

In addition to the four stages above, INDABA introduces an additional stage:

- **Task Success Evaluation**: where awards are distributed to successful team members and penalties are distributed to the unsuccessful team members. These awards are then utilised by a coordination mechanism to increase the affinity of an agent towards the allocated task.

To illustrate the cooperative problem-solving process as implemented in INDABA, an example that uses a social networks based approach for coordination through task allocation is given next. It is important to note that the social networks based approach is purposefully just briefly described as it is presented in greater detail in section 6.6. The emphasis of this example is to illustrate the five stages of the coordination process in INDABA, not the social networks based approach.

This thesis assumes that all members of the team execute the same task. Therefore, in the current implementation of INDABA, the plan formation is omitted as there is no breakdown of a task into sub-tasks. In the illustration of the role of coordination in INDABA, the emphasis is given to the potential recognition, team formation and task

success evaluation steps. The rest of this section overviews these as implemented in INDABA.

5.5.4.1 Potential Recognition

The potential recognition and task allocation functions are implemented as a simple auctioning mechanism. The auctioning mechanism awards a task to the highest bidder, i.e. the agent with the highest score. INDABA does not prescribe the mechanism to calculate the score, but for the purposes of this example it is assumed that the score is calculated using the social networks based approach (refer to chapter 6). An alternative approach, based on CNP [175] was also implemented for the purpose of simulations that are discussed in chapters 7 and 8.

During the potential recognition phase, the known task details are propagated to all agents in INDABA. For the purpose of this illustration, consider task details that consist of three parts:

- ENVIRONMENT_DETAILS, where known environment details are propagated.
- CONSTRAINT, which represents a time constraint as the maximum number of steps that each robot is allowed to execute.
- TASK TYPE, namely scout or forage.

In the initial stages, when social networks are not yet established, INDABA uses a random selection of team members. When social networks are established, INDABA caters for more complex problem domains where there is uncertainty about tasks and the suitability of each agent for a specific task. This is achieved by maintaining social networks that are based on trust and kinship, as described in chapter 6.

By means of developing and maintaining social networks, INDABA provides a mechanism for team selection optimisation, based on historical performance and trust as described in the following sections. Social networks also provide for specialisation

amongst the agents, as opposed to other frameworks that require that all agents' capabilities be known and determined upfront.

To illustrate the propagation of task details, consider that a contracting agent sends task details, as described above, to available robots, with a request to bid. If an agent is busy executing a task or prevented from executing a task (i.e. due to a malfunction), the agent will not respond to the bid. Each of the available robots evaluates its own suitability (affinity) to the task, based on its mental states.

The evaluation of task affinity is a two-step process, based on each robot's history. Firstly, the environment, as given by ENVIRONMENT_DETAILS is identified. Identification is done by encoding the environment according to its known attributes to produce an environment identifier. Secondly, the environment identifier and TASK_TYPE are used to identify the robot's previous experience in the environment identified by the environment identifier, related to the task identified by TASK_TYPE. The experience quantifier, expressed as a ratio between successful task executions and total number of task execution attempts, is then returned as the corresponding robot's bid. After all the available agents have entered a bid, teams are formed as explained in the next section. The potential recognition stage is summarised in algorithm 4.

Potential recognition

If not Auctioning agent

Receive task attributes

Identify environment

Send bid based on history and availability

Else

Send all agents task details

Collect bids

Award task to the highest bidder (team leader)

End

Algorithm 4. Potential recognition

5.5.4.2 Team Formation

The team formation algorithm has two parts, each consisting of a bidding process. The first bidding process is for selection of team leader. The robot with the highest bid is selected and awarded the responsibility of seeing the task to completion. This is either done by the team leader itself, or by a team, selected by the leader. If there are more than one robot with the same highest bid, one robot is randomly selected as the team leader.

The second bidding process is used for selection of additional team members. The additional team members are chosen according to the strength of the social link between potential team member and the team leader. In this example, the strength of social links (based on trust and kinship) in relation to the team leader is used as the bid. The team is formed by selecting the agents with the strongest bids. In a foraging example, if the carrying capacity of the agent with the highest bid is not sufficient, or there are multiple items to be collected within limited time period (as in experiments used throughout this thesis), a foraging team is formed and each member of the team has the same task to collect food (forage). The team formation process is described in p-code in algorithm 5.

Team Formation

Collect bids for team leader

Select team leader as the highest bidding agent

For all agents

    Evaluate strength of the social links between agent and team leader

End For

Select the team members according to the strength of social links

Algorithm 5. INDABA potential recognition and team formation

Once the team is selected, the task is executed. Upon completion (successful or unsuccessful), the performance of the team is evaluated.

5.5.4.3 Task Success Evaluation

The task success evaluation stage is implemented as a function that awards agents that have successfully completed their allocated tasks, with or without the help of other agents. Once task execution is completed, all the successful robots (the robots that have completed the task) are rewarded. The strength of the social links between successful agents that have participated in the team are reinforced by raising the level of trust between the successful team members. For each successful robot in the team, its affinity to the task is improved by means of increasing the ratio between the number of successful task executions and the number of attempts of task execution. The exact reward function is not prescribed, and will be problem dependent.

The next chapter explains how trust is calculated. For the purpose of illustrating the task evaluation process, let trust between two robots R_i and R_j in relation to a task T be defined as *trust* (R_i , R_j , T). The task success evaluation process is then described in algorithm 6.

Task Success Evaluation

For all Robots R_i *in team*
  If Task T successfully executed
    For all remaining Robots R_j *in team that have successfully executed Task T*
      Increase trust (R_i , R_j , T)
    EndFor
  Else
    For all remaining Robots R_j *in team that have not successfully executed Task T*
      Decrease trust (R_i , R_j , T)
    EndFor
  EndIf
EndFor

Algorithm 6. INDABA Task success evaluation

5.6 Summary

INDABA, the new MAS architecture, was presented in this chapter. INDABA consists of four layers, each of which was discussed in detail, with the emphasis on the interaction layer that encapsulates coordination mechanisms in INDABA.

The next chapter discusses the main approaches to coordination. A new approach to coordination using social networks is also presented in the next chapter.