# CHAPTER 4

## The Cognitive Actor

THE social actor, as explained in the previous chapter, is concerned with the social construction of the world around him and how this world can be represented in the mind of the actor. In other words, the actor is not only physically, but also socially and culturally embedded (or situated) in the society. On the other hand, the cognitive actor, despite the fact that it is connected with the outside world by systems of perception and action, is more concerned with internal representations and operations on those representations.

We have argued in the previous chapter that the social level can be characterised as social representations, e.g. as social structures, social constructs, and that social representations are symbols in the mind of the cognitive actor that can undergo the same operations as any other symbol or representation in the mind of the actor. In this chapter we focus on cognitive science—especially on the cognitive architecture—and leave the social actor alone. The implementation of the social construct as a representation will be described in the next chapter.

In chapter 1, we asked ourselves what kind of an actor would be capable of handling signs, relations and social constructs. In this chapter, we argue that such an actor should have a cognitive architecture, i.e. be cognitive plausible, in order to handle *representations* such as signs, relations and social constructs. We argue for the need to adopt theories of cognition and cognitive architectures. However, we neither give a complete reconstruction of the theory of cognition, nor present an extensive philosophical and in-depth cognitive discussion on what precisely cognitive architectures and their complex domain of applications are. Instead, we rely on existing theories of cognition about cognitive architectures; theories that have been thoroughly tested and have thereby proven their robustness over the last two decades.

We start with section 4.1 that discusses cognitive science as an interdisciplinary field in which cognitive architectures originate. This encompasses a dis-

cussion on levels of description. We also expose the reader to approaches such as connectionism and embodied (or situated) cognition that present different understandings of how the mind is constructed. These two approaches, even though they are claimed to be significantly different from the classical approach. We argue that their arguments are complementary rather than contradictory arguments to the classical approach. This discussion becomes apparent in our extensions of the cognitive architecture in the next chapter.

We give an overview of the cognitive architecture in section 4.2. Specifically, we consider two cognitive architectures as good candidates for implementing a cognitive plausible actor. In section 4.3, we elaborate the selection of one of these architectures[1]—ACT-R—that best suits our research purposes.

In section 4.4, we detail the internal modules and mechanisms of ACT-R, because its analysis functions as a guideline for the modelling and facilitated comprehension of a new cognitive architecture, i.e. RBot (see chapter 5).

In the final section—section 4.5—we discuss issues concerning ACT-R, and the changes or additions that we make to the principles of ACT-R in order to create a platform of cognitive actors that is suitable for our simulation demonstrated in chapter 6.

## 4.1 Cognitive science

Cognitive science is a multi-disciplinary field that draws on artificial intelligence, neuroscience, philosophy, psychology, and linguistics. The advocates of each discipline are interested in studying the brain and mind. They see the brain/mind as a complex system that receives, stores, retrieves, transforms, and transmits information (Stillings, Weisler, Chase, Feinstein, Garfield, & Rissland, 1995). Cognitive scientists claim—in opposition to the behaviourists or ecologists—that the mind is more than a simple stimulus-response mechanism, i.e. the mind is a complex construction or architecture that interprets the world and cannot be explained by physical or behavioural laws alone. The presence of *cognitive interpretation* (mechanisms) in the mind of the actor is expressed by Pylyshyn in the following statement:

> The critical aspect of the connection between stimulus conditions and subsequent behavior—the aspect that must be incorporated in the theoretical account if the latter is to capture the systematicity of the person's behavior—is that it is (a) the environment or the antecedent event *as seen or interpreted by the subject*, rather than as described by physics, that is the systematic determiner of actions; and (b) actions performed with certain *intentions*, rather than behaviors as described by an objective natural science such as physics, that enter into behavioral regularities. (Pylyshyn, 1984, p. 9)

Apparently, there is more in the mind than solely a response to signals of the environment. The mind is built up out of representations and representational schemes including "rules for building complex symbolic structures out of

[1]Such a selection process does not reject other cognitive architectures or claim that one is better than the other. It only shows which architecture conforms most to our research purposes.

simpler ones[, which] are called *combinatorial, generative or productive*" (Stillings et al., 1995, p. 5).

Algorithms, assumed to be present in the mind, produce these new symbols out of existing symbols. For instance, an addition of "9" and "8" creates a new symbol "17". However, algorithms are formal operations on symbols and are different from the representational relations between symbols and what they stand for. This difference is in the level of description, a topic that was discussed in both chapters 1, and 2. We made a distinction in several levels of description, from physical to social. In the next section, we continue the discussion about the levels of description, but for the time being leave the emphasis on *the mind of the individual*.

### 4.1.1 Levels of description

First of all, we want to recall the levels of description of Dennett (1978), discussed in chapter 2. Dennett distinguishes a rational level—a level that contains a "belief-system" in relation to the actor's goals, a functional level—a level with descriptions in the terms of functions of the system, and a physical level—a level that describes the physical properties of functional components.

Following up Dennett, many other cognitive scientists/psychologists have acknowledged that the mind cannot simply be described by a physical or rational level alone. Anderson (1990), see table 4.1, provides a nice comparison between the levels of description assigned by various cognitive scientists. Cognitive scientists are not interested much in the physical (or biological) level, because the details of the physical level are still unclear. Therefore, cognitive scientists depict the physical level as a separate layer, i.e. "the implementation layer is an approximation to the biological level" (Anderson, 1990, p. 18).

<table>
<tr><th>Dennett<br><br>(1978)</th><th>Marr<br><br>(1982)</th><th>Pylyshyn<br><br>(1984)</th><th>Rumelhart & McClelland<br>(1986)</th><th>Newell<br><br>(1982)</th><th>Anderson<br><br>(1990)</th></tr>
<tr><td>Intentional Level</td><td>Computational theory</td><td>Semantic Level</td><td></td><td>Knowledge Level</td><td>Rational Level</td></tr>
<tr><td rowspan="2">Functional Level</td><td rowspan="2">Representation and Algorithm</td><td>Algorithm</td><td>Macrotheory/ Rules</td><td>Program Symbol Level</td><td>Algorithm</td></tr>
<tr><td>Functional Architecture</td><td>Microtheory PDP models</td><td>Register Transfer Level</td><td>Implementation</td></tr>
<tr><td>Physical Level</td><td>Hardware implementation</td><td>Biological Level</td><td></td><td>Device</td><td>Biological</td></tr>
</table>

Table 4.1: Levels of description of various cognitive scientists (Anderson, 1990)[2].

Second, we should draw a distinction between the algorithm level and the implementation level, i.e. "[t]here is a distinction to be made between (a) the mental procedures and knowledge we possess that enable use to behave adap-

[2]We have added Dennett (1978) and removed Chomsky (1965) from the original table.

tively, and (b) the mechanisms that implement these procedures and knowledge" (Anderson, 1987, p. 468)[3].

The highest level in cognition is the semantic, intentional, knowledge or rational level, i.e. according to Newell, there should be another level above the symbol or algorithm level:

> *The Knowledge Level Hypothesis*. There exists a distinct computer systems level, lying immediately above the symbol level, which is characterized by knowledge as the medium and the principle of rationality[4] as the law of behavior. (Newell, 1982, p. 99)

The highest level allows one to develop psychological theories. It is not directly concerned with the algorithm or implementation level, or with what the mechanisms of the mind are about.

> Rather, it is about constraints on the behavior of the system in order for that behavior to be optimal. If we assume that cognition is optimized, these behavioral constraints are constraints on the mechanisms. This level of analysis is important, because it can tell us a lot about human behavior and the mechanisms of the mind. ...[I]t leads us to formal models of the environment from which we derive behavior. Thus, its spirit is one which focuses us on what is outside the head rather than what is inside and one which demands mathematical precision. (Anderson, 1990, pp. 22–23)

An example of a rational analysis is known from Dennett (1987)'s intentional level that ascribes beliefs, desires and intentions to actors:

> ...first you decide to treat the object whose behavior is to be predicted as a rational agent; then you figure out what beliefs that agent ought to have, given its place in the world and its purpose. Then you figure out what desires it ought to have, on the same considerations, and finally you predict that this rational agent will act to further its goals in the light of its beliefs. A little practical reasoning from the chosen set of beliefs and desires will in many—but not all—instances yield a decision about what the agent ought to do; this is what you predict the agent *will* do. (p. 17)

According to Anderson (1990), the rational analysis has some advantages. In the first place, a rational approach has the advantage of avoiding complications of the mechanistic approach.

> [First o]ne has a theory that depends on the structure of an observable environment and not on the unobservable structure of the mind...[Second, i]t offers explanation for why the mechanisms compute the way they do...[, and third, i]t offers real guidance to theory construction...[instead of relying]...on very weak methods to search a very large space of psychological hypotheses. (p. 30)

[3]This is similar to Pylyshyn (1984)'s distinction in Algorithm and Functional Architecture.

[4]For a definition of the principle of rationality, see 4.3.1: *cognitive principles*.

In studying Multi-Agent Systems (MAS), often the rational or intentional level is considered well-suited for describing (behavioural) systems, e.g. the game theory applied in economics or the rational (logical) agent (cf. Wooldridge, 2000) in MAS. In this dissertation, we pay attention to the intentional level, because we incorporate goals and interaction with the environment/social world as well. However, we also want to incorporate the functional level—algorithm and implementation levels—besides the rational or intentional level. In other words, an actor should contain (1) *representations*, (2) *cognitive mechanisms or algorithms*, and (3) *functional mechanisms*[5], because we argue that human behaviour cannot solely be explained by simplified stimulus-response behaviour. A design or a model in the form of a *cognitive architecture* is a proven methodology in unfolding the essentials of human behaviour in a more accurate way than that provided by the behaviourist approach.

The cognitive architecture is based on the fundamental view that in cognitive science an intelligent system is not completely homogeneous, i.e. it must consist of a number of functional subsystems, or modules, that cooperate to achieve intelligent information processing and behaviour. This view is supported by empirical results in cognitive psychology (Stillings et al., 1995).

This chapter primarily focuses on the classical approach of cognition. However there are also two other areas—*connectionism* and *embodied cognition*—within cognitive science that deserve attention. For instance, ACT-R, as discussed in this chapter, applies connectionism-like assumptions in its associative memory. In the case of embodied cognition, we adopt the ideas of the subsumption (architecture) of behaviours for constructing the awareness of social situations.

### 4.1.2 Connectionism and embodied cognition

We will explain shortly what the *connectionist approach* means as regards the construction of a cognitive architecture. The connectionist approach originates from the PDP[6] group (Rumelhart et al., 1986) and presents a quite different approach from that of the classical cognitive approach; it even seeks to replace it.

A connectionist architecture consists of a network of computationally simple processing *units* or *nodes* that are interconnected with each other and are capable of transporting only simple *signals*[7]. A short description of connectionism is given by Fodor and Pylyshyn (1988, p. 5):

> Connectionist systems are networks consisting of very large numbers of simple but highly interconnected "units". Certain assumptions are generally made both about the units and the connections: Each unit is assumed to receive real-valued activity (either excitatory or inhibitory or both) along its input lines. Typically the units do little more than sum this activity and change their state as

[5]These three requirements are present at respectively, the rational (knowledge) level, the algorithm level, and the implementation level (or functional architecture).

[6]Parallel Distributed Processing

[7]We refer to the previous chapter for the definition of *signals* and how they differ from *signs*.

> a function (usually a threshold function) of this sum. Each connection is allowed to modulate the activity it transmits as a function of an intrinsic (but modifiable) property called its "weight". Hence the activity on an input line is typically some non-linear function of the state of activity of its sources. The behavior of the network as a whole is a function of the initial state of activation of the units and of the weights on its connections, which serve as its only form of memory.

The attraction to connectionist networks is due to their similarity to neural networks. Neurons have several inputs that can trigger a neuron to fire, similar to the threshold level of a unit of a connectionist model; connections between the units tend to have a structure that is similar to the structure of connections between axons and dendrites. However, in order for a connectionist model to constitute a theory of a cognitive architecture, it has to make claims about *representations* and processes that occur at the semantic or knowledge level. Symbolic computation is something that is lacking in a connectionist model[8]. For instance, it is difficult for connectionist models to reason about the difference between 'John loves Mary' and 'Mary loves John'. According to classical cognitive scientists, it can be argued that connectionist models are suitable for lower levels, such as the latency of production firing and retrieval of information from memory.

We give attention to connectionism here, because in hybrid architectures, like ACT-R, activation as a property of *symbolic chunks* is similar to activation of a unit in connectionist models. A connectionist model has the ability to spread activation over time. In other words, the activation of a unit is spread over other units that are connected to it. This also occurs in classical models that borrow properties of connectionist models.

*Embodied Cognition*[9] (EC), part of the so-called New AI, is an opponent of GOFAI stating that an actor is not dependent on internal representations, but is a more or less behavioural system situated in and responding to its environment:

> Instead of emphasizing formal operations on abstract symbols, [EC] focuses attention on the fact that most real-world thinking occurs in very particular (and often very complex) environments, is employed for very practical ends, and exploits the possibility of interaction with and manipulation of external props. It thereby foregrounds the fact that cognition is a highly *embodied* or *situated activity* and suggests that thinking beings ought therefore be considered first and foremost as acting beings. (Anderson, 2003, p. 91)

Rodney Brooks has had the most impact in the field of New AI. He published a collection of papers in his book *Cambrian Intelligence* (Brooks, 1999) that covers many aspects of EC and the field of robotics. In one of those papers, "Intelligence without Representation", Brooks takes a radical position against classical AI (Brooks, 1991b, p. 140):

[8] Fodor and Pylyshyn (1988) provide strong arguments as to why connectionist models cannot replace classical models and that most of them concern implementation issues.

[9] Within Cognitive Science, Embodied Cognition falls in the category of New AI, into which we can classify Embodied Action, Situated Cognition or Situated Action as well.

> We have reached an unexpected conclusion (C) and have a rather radical hypothesis (H).
>
> **(C)** When we examine very simple level intelligence we find that explicit representations and models of the world simply get in the way. It turns out to be better to use the world as its own model.
>
> **(H)** Representation is the wrong unit of abstraction in building the bulkiest parts of intelligent systems.
>
> Representation has been the central issue in artificial intelligence work over the last 15 years only because it has provided an interface between otherwise isolated models and conference papers.

Similar to connectionism, this statement attacks *the existence of representations* in a manner equivalent to the debate that the connectionists have postulated against representations. However, this attack is from the outside rather than the inside, i.e. representations prevail in the world and not in the mind.

The counterclaim of classical AI is that an intelligent organism cannot reason without representations or a physical symbol system. Kirsh (as cited by Anderson, 2003) states that:

> ...[I]f a task requires knowledge about the world that must be obtained by reasoning or by recall, rather than by perception, it cannot be classified as situation determined. Principle candidates for such tasks are:
>
> - Activities which involve other agents, since these often will require making *predictions* of their behaviour.
> - Activities which require response to events and actions beyond the creature's current sensory limits, such as taking precautions now for the future, avoiding future dangers, contingencies, idle wandering—the standard motive for internal lookahead.
> - Activities which require understanding a situation from an objective perspective such as when a new recipe is followed, advice is assimilated, a strategy from one context is generalized or adapted to the current situation. All these require some measure of conceptualization.
> - Activities which require some amount of problem solving, such as when we wrap a package and have to determine how many sheets of paper to use, or when we rearrange items in a refrigerator to make room for a new pot.
> - Activities which are creative and hence stimulus free, such as much of language use, musical performance, mime, self-amusement.
>
> These activities are not isolated episodes in a normal human life. Admittedly, they are all based on an underlying set of reliable control

> systems; but these control systems are not sufficient themselves to organize the *global* structure of the activity. (Kirsh, 1991, pp. 173–174)

Although we also think that Brooks' hypothesis and conclusion are too strong, we agree with the following key aspects that characterise his *behaviour-based* style of work (Brooks, 1991a, 1999, pp. 138–139):

**Situatedness** The [actors] are situated in the world—they do not deal with abstract descriptions, but with the here and now of the world directly influencing the behavior of the system.

**Embodiment** The [actors] have bodies and experience the world directly—their actions are part of a dynamic with the world and have immediate feedback on their own sensations.

**Intelligence** They are observed to be intelligent—but the source of intelligence is not limited to just the computational engine. It also comes from the situation in the world, the signal transformations within the sensors, and the physical coupling of the [actor] with the world.

**Emergence** The intelligence of the system emerges from the system's interactions with the world and from sometimes indirect interactions between its components—it is sometimes hard to point to one event or place within the system and say that is why some external action was manifested.

These aspects are in line with the approach taken by Brooks to decompose an actor control problem. This is carried out in task achieving behaviours such as avoiding objects, wandering, exploring, building maps, monitoring changes etc. rather than in *functional units* such as perception, modelling, planning, task execution, and motor control. The idea of Brooks was to design a completely autonomous mobile robot that would perform many complex information processing tasks in real time. This lead to the *subsumption architecture*, see figure 4.1, that broke down an actor into vertical modules, each addressing a small part of the total behaviour (Brooks, 1986).

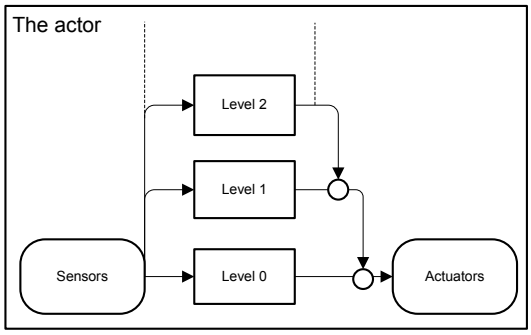


Figure 4.1: The subsumption architecture (Brooks, 1986).

The subsumption architecture is constructed in the following way. As a first step, you build a system at level 0 that is stable, and assign it as *level 0 competence*. Another layer that is able to influence behaviour at the lower level, called the first level control system, is added to this layer. In this way many levels of behaviour can be built on top of each other, which produce a complex system of (hard-wired) behaviours.

For example, assume a robot on Mars[10] that has the main goal of creating a map of the surface (competence level 2) with help of exploring that is done by looking for places that are reachable (competence level 1). And finally, the lowest level in the system (level 0 competence) makes sure that the robot survives by evading objects. The organisation of levels allows the lower levels to overrule or inhibit the higher levels. In this way, the lower levels act as fast-responding mechanisms (reflexes) while the higher levels determine the direction where to go and try to fulfil the overall (main) goal.

In the next chapter, we integrate embodied or situated cognition, and adopt the principles of the *subsumption* architecture. We argue not only that the subsumption architecture is suitable for detecting physical changes in the environment and therefore creates the notion of *physical* situatedness, but that the same principles can be applied to creating an actor that responds to *social* changes (changing social constructs) and thereby is *socially* situated. We close the discussion about embodied cognition here with the claim that a cognitive architecture should have representations and a physical symbol system, and that embodied or situated cognition can support the symbolic cognitive architecture in responding better to changes in the environment.

In the following sections, we focus our discussion on the cognitive architecture and the selection of such an architecture. The selected architecture will form the core of our actor—RBot—that is discussed in the next chapter.

## 4.2 The Cognitive Architecture

In the previous chapter we argued that an organisation consists of a collection of actors that have their own mental representation of the organisation as a whole. Such actors are in need of a cognitive model that supports representations and operations on those representations. A cognitive architecture implements such a model, and gives the actors the ability to maintain beliefs, desires and intentions (goals) at the rational level, and to manipulate representations at the algorithm level. They also have functional mechanisms (e.g. memory storage and retrieval functions) at the implementation level, a level that is an approximation of the physical level.

Taatgen (1999, p. 28) defines an architecture of cognition as an algorithm that simulates a non-linear theory of cognition. This algorithm can be used to make predictions in specific domains and for specific tasks, see figure 4.2. Hence, the simulation of cognitive phenomena consists of, apart from the architecture, a specific task domain (or environment) that needs to be implemented, combined

[10]For the interested reader, see also Brooks' paper "Fast, cheap and out of control: a robot invasion of the solar system" (Brooks & Flynn, 1989).

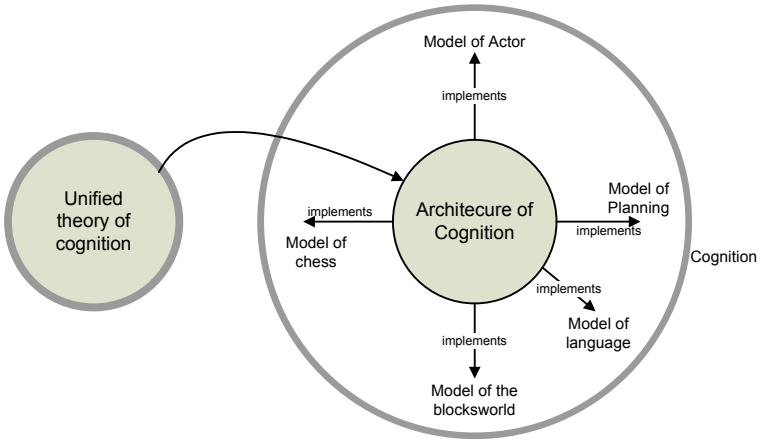


Figure 4.2: Relationship between theory, architecture, models, actor and cognition (adapted from Taatgen, 1999).

with knowledge necessary to complete the tasks in that domain.

A *cognitive architecture* is a theory about the computational structure of cognition (Anderson & Lebiere, 1998; Newell, 1990; Pylyshyn, 1984), i.e. it specifies a set of processes, memories, and control structures that are capable of encoding content and executing programs (Lewis, 2001). The two mainstream cognitive architectures[11] in artificial intelligence are SOAR (Laird et al., 1987) and ACT-R (Anderson & Lebiere, 1998). A choice between one of these architectures seems to be the most economical and logical one for our cognitive actor[12].

### 4.2.1 SOAR and ACT-R

In this section, we give a short summary of the cognitive architectures SOAR and ACT-R. Some details of the cognitive plausible actor were already introduced in chapter 2. These encompassed the components or properties of perception, (working) memory, cognition, learning, action, motor-control commonly found in cognitive architectures, like SOAR and ACT-R.

#### 4.2.1.1 Summary of SOAR

The implementation of a cognitive architecture started with Newell and Simon (1972) who introduced a production-system theory of human cognition. Newell tested a number of different production systems, concluding with his SOAR theory of human cognition (Newell, 1990). The SOAR theory posits that cognitive behaviour has at least the following characteristics: it (1) is goal oriented; (2) takes place in a rich, complex, detailed environment; (3) requires large

[11] EPIC (Meyer & Kieras, 1997) and 3CAPS (Just & Carpenter, 1992) are examples of other cognitive architectures not discussed here. For a discussion, see Taatgen (1999).

[12] Although the cognitive actor is part of a multi-actor system, the emphasis here is not on MAS but on the cognitive architecture of the actor.

amounts of knowledge, the use of symbols and abstractions; (4) is flexible, and a function of the environment; and (5) requires learning from the environment and experience (Lehman et al., 2006). The theory and properties of cognitive behaviour have been the requirements for developing the cognitive model or architecture SOAR.

SOAR[13] (**S**tate, **O**perator **A**nd **R**esult) is a *cognitive* or *problem solving architecture* in which problem solving occurs with the help of tasks accomplished in problem spaces, i.e. the *problem space hypothesis*[14]. SOAR solves tasks in the following way:

> To realize a task as search in a problem space requires a fixed set of *task-implementation* functions, involving the retrieval or generation of: (1) problem spaces, (2) problem space operators, (3) an initial state representing the current situation, and (4) new states that result form applying operators to existing states. To control the search requires a fixed set of *search-control* functions, involving the selection of: (1) a problem space, (2) a state from those directly available, and (3) an operator to apply to the state. Together, the task-implementation and search-control functions are sufficient for problem space search to occur. (Laird et al., 1987, p. 11)

SOAR has an architecture—see figure 4.3—that consists of five main parts:

1. A *Working Memory* that holds the complete processing state for problem solving in SOAR and exists out of three components: (1) a context stack that specifies the hierarchy of active goals, problem spaces, states and operators; (2) objects, such as goals and states; and (3) preferences that encode the procedural search-control knowledge, i.e. structures that indicate the acceptability and desirability of objects.

2. A *Decision Procedure* that examines the preferences and the context stack, and changes the context stack, i.e. it influences what goal or action is put on top of the stack.

3. A *Working Memory Manager* that maintains and cleans up the working memory from elements that are not needed anymore.

4. A *Production Memory* which is a set of productions that can examine any part of working memory, add new objects and preferences, and augment existing objects, but cannot modify the context stack.

5. A *Chunking Mechanism* is a learning scheme for organising and remembering ongoing experience automatically. New chunks are created, when

[13] A comprehensive and detailed explanation of SOAR can be found in (Laird et al., 1987; Lehman et al., 2006; Lewis, 2001; Newell, 1990) or at http://sitemaker.umich.edu/soar.

[14] "The rational activity in which people engage to solve a problem can be described in terms of (1) a set of states of knowledge, (2) operators for changing one state into another, (3) constraints on applying operators and (4) control knowledge for deciding which operator to apply next" (Newell, 1990, p. 33)

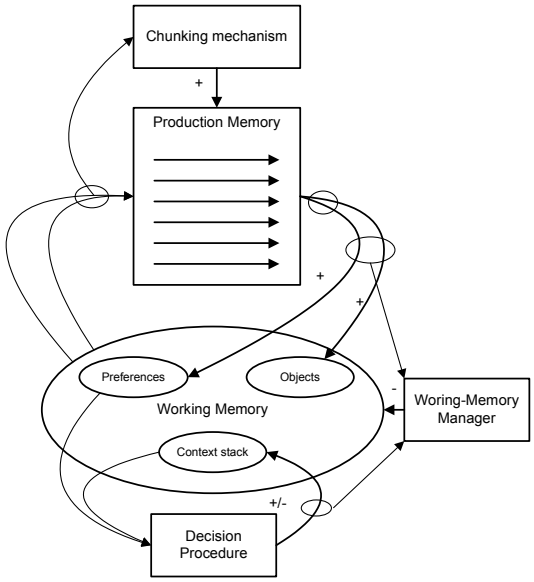


Figure 4.3: Architectural structure of SOAR (Laird et al., 1987).

> SOAR encounters an insolvable problem or *impasse*. The impasse causes SOAR to create a subgoal, which is created to find the necessary knowledge. The chunking mechanism stores away the knowledge so that under similar circumstances in the future, the knowledge will be available.

The complex mechanism behind SOAR works as follows: (1) a certain problem is projected onto a problem space; (2) the representation of the problem and its context objects is changed by adapting the current problem space to a new one; (3) the replacement is controlled by the *decision cycle* that consists of (a) the *elaboration phase*—new objects, new augmentations of old objects, and preferences that are added to working memory, and (b) the *decision procedure* that either replaces an existing object or creates a subgoal in response to an impasse upon which the next cycle starts. The general behaviour of SOAR is a divide-and-conquer approach oriented towards a goal, i.e. during its search—with the help of the decision cycle(s)—it will make use of subgoals to solve sub-problem spaces that will contribute to solving the main goal.

#### 4.2.1.2 Summary of ACT-R

The ACT-R (**A**daptive **C**ontrol of **T**hought, **R**ational) architecture (Anderson & Lebiere, 1998) is a production system similar to SOAR, but with many differences regarding its implementation. ACT-R (Anderson, 1990; Anderson & Lebiere, 1998; Anderson, Bothell, Byrne, Douglass, Lebiere, & Qin, 2004) consists of a set of modules: perception and motor (visual and manual module) that connect ACT-R with the outside world, a declarative memory retrieving information from memory, a memory containing productions, and a goal module

for keeping track of current goals and intentions. The coordination and behaviour of these modules is controlled by a production system, similar to SOAR. In addition, ACT-R has a set of buffers in between the modules and the production system. The contents of the buffers is regulated by separate processes that are active in the modules, i.e. there is (some) independency caused by separate systems that determine the contents of the buffer. For instance, the contents of the visual buffer represent the products of complex processes of the visual perception and attention systems. Furthermore, the decision or "critical" cycle in ACT-R is similar to that in SOAR: "the buffers hold representations determined by the external world and internal modules, patterns in these buffers are recognized, a production fires, and the buffers are then updated for another cycle" (Anderson et al., 2004, p. 1038).

ACT-R's distinction from SOAR lies not so much in its processing structure at the algorithm level, but at the implementation level or in its functional architecture. Learning does not only unfold, through the creation of new chunks[15] by solving (sub)goals, as that in SOAR, but occurs also at the sub-symbolic level. Hence, learning occurs not only by analogies and creation of new chunks as the result of production firing in the decision cycle, but also at the sub-symbolic level of productions and chunks. Every production and chunk maintains an activation level that increases with successful use, which leads to faster retrieval of chunks and production matching and execution. The combination of sub-symbolic activation with symbolic chunks makes ACT-R a hybrid architecture[16].

## 4.3 Comparing SOAR and ACT-R: selecting a cognitive architecture

In this section, we compare the properties or characteristics of both architectures for similarities, followed by a more in-depth discussion of the differences to inform our selection of one of the architectures[17].

First, we look at a set of constraints. Newell (1980, 1990) described the mind as a solution to a set of functional constraints (e.g. exhibit adaptive (goal-oriented) behaviour, use of language, operate with a body of many degrees of freedom) and a set of constraints on construction (a neural system growing by evolutionary processes, arising through evolution) (Lewis, 2001).

Both architectures, SOAR and ACT-R, conform to three functional constraints a) exhibiting flexible, goal-driven behaviour, b) learning continuously from experience, and c) exhibiting real-time cognition. However, ACT-R also conforms to the constraint on construction by implementing sub-symbolic properties that exhibit a connectionist-like neural network.

Second, in line with Lewis (2001), SOAR and ACT-R both implement the five major technical principles in cognitive science:

[15] Chunks in SOAR are productions, while in ACT-R they are facts or declarative chunks.

[16] Hybrid architectures are a mix of connectionism (neuron-like networks) and symbolism (physical symbol system).

[17] The comparison and selection are mainly based on ACT-R version 4.0 and the SOAR version without episodic and semantic memories.

1. The *physical symbol system hypothesis* that assumes that only physical symbol systems support intelligent, flexible behaviour. The system is required to process, manipulate and compose *symbols and symbol-structures* to support intelligent, flexible behaviour.

2. A *cognitive architecture*: a fixed structure that processes the variable content and executes programs specified by a set of processes, memories and control structures. Different cognitive models can be developed for a specific task environment (problem-space) with the help of a single architecture. What is of importance is the structure of the architecture and the cognitive model's program and *not* the implementation in a programming language. In other words, theoretical claims are made about the model and not about the underlying programming language.

3. A *production system*, consisting of productions (condition-action pairs) that are matched to patterns in the declarative or working memory, which in turn create state changes in the memory. The behaviour of the system is *goal-oriented*, i.e. the main goal is solved by creating subgoals (divide-and-conquer) when a goal cannot immediately be solved.

4. Cognition in both SOAR and ACT-R is achieved by a search in problem spaces supported by a three stage (recognise, decide, act) automatic/deliberate control structure. The difference between both architectures is that SOAR fires all procedures in parallel[18] and matches those to the current state or goal[19]—resulting in a collection of (different) operators and end-states, after which SOAR decides which state is consistent and acts based on that state. On the other hand, ACT-R first selects only the procedures that match with the current goal, selects between those matching procedures (conflict resolution), and fires/enacts only the selected procedure which results in state change(s). Both architectures have a so-called impasse (SOAR) or conflict resolution mechanism (ACT-R) that solves conflicts for SOAR when it is not clear what needs to be done next, i.e. when there are several operators and end-states that are all consistent at the same time, and for ACT-R when there are two or more productions matching at the same time. In the case of SOAR, a new subgoal is created by a decision-procedure, while in ACT-R, the conflict resolution forces the selection of a single procedure based on the success ratio built up from experience in the past.

5. SOAR and ACT-R both have learning processes. SOAR learns by generating new productions in memory by preserving the results of previous problem solving that occurs in response to an impasse. ACT-R, on the other hand, learns not only by creating new declarative chunks in mem-

[18] Procedure and production are used interchangeably as well as in most literature, but they refer to the same phenomena.

[19] The same for state and goal; they are interchangeable, in so far that goal and state are both reacting for executing an action.

ory, but also by production compilation[20] that creates new productions by compiling facts from declarative memory into new productions. SOAR, as well as ACT-R, create faster execution of productions, by recording previous successful paths of solutions. Apart from this, ACT-R applies subsymbolic learning properties to chunks and procedures that give it a finer grain than SOAR. This application also gives ACT-R's architecture the features of a neural network.

In order to select a cognitive architecture, specific selection criteria need to be met. Over and above the constraints and the technical principles, the architectures need to be capable of implementing (or serve as a plug-in into) a multi-actor or computational organisation model based on cognitive plausible actors that are able to coordinate and cooperate to fulfil organisational goals.

### 4.3.1 Selection of a cognitive architecture[21]

In our research, the selection process requires criteria that determine whether a cognitive architecture is suitable for implementing a computational organisation model. The cognitive architecture not only has to fulfil (a) the cognitive demands to create a cognitive plausible actor, but also (b) the social and organisational demands that are needed to support organisational behaviour.

First, we will address the cognitive principles and the differences between SOAR and ACT-R. Then we will introduce the requirements of a cognitive plausible actor for it to conform to the concept of bounded rationality: "the concept that Simon (1945, p. 80) coined to conceptualize the limitations of "perfect" representations of the environment by the human actor and also of the mental processing capacity of the actor" (Helmhout et al., 2004, p. 9). The last requirement concerns the social and organisational demands. We will adopt requirements from computational mathematical organisation theory: the ACTS theory to address these demands (Carley & Newell, 1994).

**Cognitive Principles**

According to Anderson (1990), a cognitive architecture, such as SOAR and ACT-R, is defined along the following principles:

1. *Cognitive Impenetrability* (Pylyshyn, 1984; Anderson, 1990, p. 11): "The operations at the functional architecture level are not affected by the organism's goals and beliefs".

   > Although the architecture might vary as a function of physical or biochemical conditions, it should not vary directly or in logically coherent or in rule-governed ways with changes in the

[20] In SOAR, there is only a production memory. In ACT-R there is also a declarative memory: ACT-R creates new chunks in declarative memory; the creation of new productions (production-compilation (Taatgen, 1999)) is present in the newer version ACT-R 6.0.

[21] The selection of the architecture was made based on the characteristics of SOAR and ACT-R in the year 2003.

> content of the organism's goals and beliefs. If the functional architecture were to change in ways requiring a cognitive rule-governed explanation, the architecture could no longer serve as the basis for explaining how changes in rules and representations produce changes in behavior. Consequently, the input-output behavior of the hypothesized, primitive operations of the functional architecture must not depend in certain, specific ways on goals and beliefs, hence, on conditions which, there is independent reason to think, change the organism's goals and beliefs; the behavior must be what I refer to as *cognitively impenetrable*. (Pylyshyn, 1984, p. 114)

For instance, in ACT-R, the sub-symbolic level is not logically coherent or rule-regulated, i.e. only the symbolic level is influenced by the semantic contents of our knowledge (Anderson, 1990).

2. *Physical Symbol System Hypothesis* (Newell, 1980, p. 170):

   > The necessary and sufficient condition for a physical system to exhibit general intelligent action is that it be a physical symbol system.
   >
   > **Necessary** means that any physical system that exhibits general intelligence will be an instance of a physical symbol system.
   >
   > **Sufficient** means that any physical symbol system can be organized further to exhibit general intelligent action.
   >
   > **General intelligent action** means the same scope of intelligence seen in human action: that in real situations behavior appropriate to the ends of the system and adaptive to the demands of the environment can occur, within some physical limits.

3. *Principle of Rationality* (Newell, 1982): "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action" (p. 102). According to Newell, this principle is part of the Knowledge Level Hypothesis, thereby stating that the "principle of rationality does not have built into it any notion of optimal or best, only the automatic connection of actions to goals according to knowledge" (Newell, 1982, p. 102). In a similar vein, Anderson (1990) states: "there are three terms related by the principle of rationality—goals, knowledge, and behavior. Given any two, one can infer or predict the third. Thus, if we know a person's goals and observe his behavior, we can infer his knowledge" (p. 15).

4. *General Principle of Rationality* (Anderson, 1990, p. 28): "The cognitive system operates at all times to optimize the adaptation of the behavior of the organism". Anderson avoids the use of knowledge and defines a more abstract principle of rationality, by stating that the cognitive system is optimising the adaptation of the behaviour at all times, i.e. the system is continuously trying to be in coherence with itself and with its environment.

> [A] principle of rationality should be defined in terms of a person's experience and not knowledge. The implications of experience are uncertain and fundamentally probabilistic, whereas the implications of knowledge are certain. Because it is experience that people have direct access to and not knowledge, our analysis of what behavior is rational has to face up to the probabilistic structure of uncertainty. (Anderson, 1990, p. 17)

**Differences between SOAR and ACT-R**

The first *difference* between SOAR and ACT-R is that SOAR operates more at the knowledge level (principle of rationality), while ACT-R operates at the level of optimising the adaptation of the behaviour (also at the sub-symbolic level). Hence, there is a difference in the *theoretical assumptions* between SOAR and ACT-R. The second difference is the *sequential control* of both architectures. Although both architectures do focus on a single goal, the procedures in SOAR can fire in parallel, while the procedures of ACT-R fire in serial. Third, there is a difference in the structure of *long-term memory*: ACT-R distinguishes facts (declarative memory) from productions (production memory) while SOAR only maintains productions in long-term memory. Furthermore, the *learning capabilities* of SOAR are based on two learning mechanisms[22] 'chunking' and reinforcement learning (the adjustment of values of operators). On the other hand, ACT-R has learning at the sub-symbolic level, production compilation (ACT-R version 6.0), history of experience of procedures, and creation of declarative facts. And finally, the *latency derivation* (the time it costs to retrieve and execute) is implemented in a different way for both architectures. While for SOAR there is a constant latency, ACT-R calculates latency based on activation of the needed elements in memory. Table 4.2 gives an overview of the differences between SOAR and ACT-R.

**Bounded rationality**

Another important criterion for accepting an architecture is the 'bounded rational' nature of the model. This is a term used to designate rational choice that takes into account ecological and cognitive limitations of both knowledge and computational capacity. This relates to the question of how an organism or cognitive architecture makes inferences about unknown aspects of the environment. Gigerenzer and Goldstein (1996, p. 650) state that there are two views con-

[22]The most recent developments show that SOAR has now four architectural learning mechanisms: (1) Chunking automatically creates new rules in LTM whenever results are generated from an impasse. It speeds up performance and moves more deliberate knowledge retrieved in a substate up to a state where it can be used reactively. The new rules map the relevant pre-impasse WM elements into WM changes that prevent an impasse in similar future situations. (2) Reinforcement learning adjusts the values of preferences for operators. (3) Episodic learning stores a history of experiences, while (4) semantic learning captures more abstract declarative statements (Lehman et al., 2006). Apparently architectures are always 'under construction'. For instance, there is a major change in the latest release of SOAR: the introduction of an episodic and a semantic memory, i.e. the latest version of SOAR assumes the existence of different types of memory besides the production memory as well.

| ACT-R | SOAR |
|---|---|
| *Foundational assumptions* | |
| - Mechanisms should be implemented in neural-like computation<br>- Cognition is adapted to the structure of the environment (Anderson, 1990)<br>- General principle of rationality | - Humans approximate knowledge level systems<br>- Humans are symbol systems<br>- Principle of rationality |
| *Sequential control* | |
| - Single goal hierarchy<br>- Goals are specific knowledge chunks<br>- Adaptive, satisficing conflict resolution on production | - Single goal hierarchy<br>- Goals are states created by architecture<br>- Knowledge-based, least commitment conflict resolution on operator |
| *Long Term Memory (LTM)* | |
| - Distinctive buffers: declarative(facts) and procedural memory(productions)<br>- Declarative memory is a network with activations and associative strengths | - Uniform LTM: all LTM in procedural form<br>- Graph structure storing procedures |
| *Learning* | |
| - The strength of declarative and procedural memory increases as a power function of practice and decrease as a power function of delay<br>- Procedural knowledge is tuned, based on experience<br>- Procedural knowledge is acquired by production compilation (version 6.0)<br>- Declarative knowledge is acquired through internal production firing and external perception<br>- Activations and associative strengths of declarative memory are acquired through experience | - All long-term knowledge arises through chunking<br>- Reinforcement learning<br>- New procedures are created with the help of chunking |
| *Latency derivation* | |
| - Latencies are derived from the sum of times for each chunk to be matched to each condition in the rule.<br>- Depends on the strength of a rule and the strength of the memory elements it matches | - Constant latency is per recognize-decide-act cycle<br>- Total latency depends on number of decision cycles plus the time for external actions |

Table 4.2: Comparison of SOAR and ACT-R (Johnson, 1997).

cerning this question. The first is the "...classical view that the laws of human inference are the laws of probability and statistics...Indeed, the...probabilists derived the laws of probability from what they believed to be the laws of human reasoning (Daston, 1988)". The other view "...concluded that human inference is systematically biased and error prone, suggesting that the laws of inference are quick-and-dirty heuristics and not the laws of probability (Kahneman, Slovic, & Tversky, 1982)". According to Gigerenzer and Goldstein, there is a third way of looking at inference that focuses on "the psychological and ecological rather than on logic and probability theory" (p. 651). This view is the proposal of bounded rationality models instead of the classical rationality models.

Simon (1945, 1956, 1982) stated that human beings are boundedly rational, and that they rather 'satisfice' than optimise. The fundamental characteristics of bounded rationality are the following, which can be incorporated into a cognitive architecture:

1. Limitation on the organism's ability to plan long behaviour sequences, a limitation imposed by the bounded cognitive ability of the organism as well as the complexity of the environment in which it operates.

   Both SOAR and ACT-R work with temporal mappings of declarative knowledge and productions that are active in the current state of the actor. Only a certain set of operators or productions that are active (above a threshold) can be matched with the current state or goal, limiting the actor's possibilities. At the same time, no deterministic mathematics can be applied to identify the next state, or the many that follow after that state. ACT-R adds even more to bounded rationality, i.e. activation levels can drop below the threshold value resulting in 'forgetting' or memory decay.

2. The tendency to set aspiration levels for each of the multiple goals that the organism faces.

   Both SOAR and ACT-R set preferences to expected end-states or procedures (utilities of operators or procedures) that match the current state rather than to goals. Both SOAR and ACT-R lack an explicit intentional module that enables deliberation over goals at the *rational* level.

3. The tendency to operate on goals sequentially rather than simultaneously because of a "bottleneck of short-term memory".

   This argument holds for both architectures. SOAR initially operates as a parallel architecture, because all possible productions fire in parallel, while ACT-R only fires the selected procedure. However, in due course, SOAR selects (possible by impasse) the preferred end-state, ending up with the outcome of one procedure. This *recognize-decide-act cycle* gives SOAR and ACT-R their serial sequence and causes for both a serial/cognitive bottleneck when handling goals (cf. Lehman et al., 2006; Taatgen, 2005; Taatgen & Anderson, 2006).

4. 'Satisficing' rather than optimising search behaviour.

   The actor creates behaviour that 'satisfices' its own preferences, which is most of the time not the optimal economical behaviour to solve solutions for itself or the environment (including other actors). This is due to (1) the *non-deterministic* nature of both architectures, (2) the incomplete and imperfect mental representation of the surroundings, (3) the learning based on actor's own experiences and, (4) particularly in the *case of ACT-R*, lowering activations of stored knowledge (associative memory at the sub-symbolic level) and strengthening it on retrieval.

**Social and organisational demands**

The cognitive aspects of an architecture discussed up to this point are important for verifying the cognitive plausibility of the architecture. The previous chapter had highlighted the need for an actor to exhibit social stable behaviour in the form of, for instance, communicating with others, maintaining representations of other actors and coordinating tasks and actions with others. The ACTS theory[23] (Carley & Prietula, 1994b) addresses the issue of (shared) tasks and social situatedness by creating an extension of the original model of bounded rationality by

> ... (1) replacing [or extending] the general principles of the boundedly rational agent with a full model of the cognitive agent that exhibits general intelligence; and (2) replacing general notions of environmental constraints with detailed models of two environmental components—the task and the organizational social-situation within which the task and agent are situated. (p. 55)

In SOAR and ACT-R, the modelling of a task (problem-space) is done with the help of goal-directed behaviour, but the organisational setting requires SOAR and ACT-R to understand the (shared) task and social environment to constitute an 'organisational' cognitive architecture. The task and the social axiom of ACTS theory define that such an organisational agent performs one or more tasks in an organisation to achieve specific personal, task and/or social (group and organisational) goals, and that it occupies a (formal and informal) position in the organisation that is a socio-cultural-historical artefact involving one (or more) socially-situated roles. Such an organisational setting can only be created when a cognitive plausible actor is part of a task environment or multi-actor system.

In a MAS, a cognitive actor needs the capability to communicate and exchange instructions (speech) with other actors. Therefore, it is necessary to have a perception and motor module for each actor. The perception and motor define the border for the cognitive architecture. Beyond the modelling of the individual cognition by drawing on cognitive science, one needs to consider, as explained in chapter 3, arguments within social psychology, sociology and organisational theory.

From among existing implementations, SOAR is the only architecture that has some applications in multi-actor systems, like Plural-SOAR (Carley, Kjaer-Hansen, Newell, & Prietula, 1991) or TASCCS (Verhagen & Masuch, 1994). Therefore, it promises to be a good candidate for modelling organisational behaviour.

However, ACT-R cannot be easily outranked despite the fact that it does not have many implementations of multi-actor simulations. It, nonetheless, has a perception and motor module making communication with the outside world feasible. Therefore, it is possible to create a multi-actor simulation without much effort. In addition, ACT-R's learning capabilities, sub-symbolic level properties

[23] ACTS theory: "... organizations are viewed as collections of intelligent Agents who are Cognitively restricted, Task oriented, and Socially situated." (Carley & Prietula, 1994b, p. 55).

and the empirically tested and proven internal workings of ACT-R pose it as a strong candidate as well.

Essential to both architectures is that they can be implemented as multi-actor systems. They have been verified as appropriate candidates for cognitive psychological experiments, most of which are based on interaction with the outside world (requiring perception and motor). It is this capability of interaction with the outside world that makes both architectures suffice in creating multi-actor systems. However, ACT-R with its sub-symbolic properties and empirical validation comes closer to the neural (neurally plausible) system, and gives a finer grain to the cognitive architecture with its more sophisticated activation/latency mechanism. Johnson (1997) found two problem areas for SOAR: (1) the difficulty of SOAR to account for probabilistic move selection, and (2) the independent efforts of history and distance to goal on the likelihood of executing a move. Whereas ACT-R's control mechanism appears to be (1) well-supported by empirical data, (2) specifies the order and latency of control, and (3) is capable of dealing with complex cognitive phenomena. Irregardless of the proven multi-actor capability of SOAR, we adopt the ACT-R architecture in this research based on the cognitive criteria. Nonetheless, we will draw on the useful processes and component structure of SOAR. For instance, the graph structure of productions in SOAR—compared with ACT-R's list structure—provides better control and indexing of associations in memory.

## 4.4 ACT-R

The selection process has resulted in a preference for ACT-R as a model or blueprint for the implementation of a new cognitive actor architecture RBot[24], which will be discussed in the next chapter. In the remaining part of this chapter, we explain the inner workings of ACT-R in more detail.

In the last 15 years, production systems such as ACT-R (Anderson & Lebiere, 1998), 3/4CAPS (Just & Carpenter, 1992), EPIC (Meyer & Kieras, 1997), CLARION (Sun, 2003), and SOAR (Newell, 1990) have begun to truly realise the potential that Newell (1973, pp. 463–464) has ascribed to them.

The ACT-R architecture can be singled out as the one that comes closest to meeting all *levels of description* (Dennett, 1978, 1987, 1991; Gazendam & Jorna, 2002; Anderson, 1990; Jorna, 1990) of a (cognitive) actor. It can be described by three or four levels of description to predict the behaviour of the overall system, i.e. the implementation level as an approximation of the physical level (the sub-symbolic level), the functional level—the functionality of procedures interacting with declarative chunks, and the intentional level—the goals of the system or intentional module. ACT-R is a production system that specifies processes of perceiving, retrieving, storing and using information or chunks of knowledge. The three important components of ACT-R are: "[1] *rational analysis* (Anderson, 1990) [, see section 4.4.1], [2] the distinction between *procedural* and *declarative* memory (Anderson, 1976), and [3] a *modular* structure in which components

[24] In chapter 5, we implement RBot as well as a multi-actor environment, the so-called MRS (Multi-RBot System).

communicate through *buffers*" (Taatgen & Anderson, 2006, p. 7).

The declarative memory contains facts and associations between facts; the "know-what". The procedural memory contains procedures (condition-action rules) that react on patterns of knowledge stored as symbols or expressions in data-structures; the "know-how".

The modular structure creates high level independency between different types of cognitive processes. In other words, in ACT-R, independency is created with help of buffers. Communication or interaction between modules is only possible through these buffers.

Figure 4.4 presents the basic architecture of ACT-R version 5.0[25], which shows the modularity and is composed of the following: the goal/intentional module that keeps track of the goals and intentions, the declarative module for retrieving information, the manual (motor) module and the visual (perception) module. The central production system controls the interaction between the modules and reacts on patterns that are currently active in the buffers of the modules.

ACT-R's central production system contains a recognition, selection and execution cycle: recognition is implemented in ACT-R by a condition-matching function in which a goal is matched to a procedure; the selection of a potential procedure is done with help of conflict resolution[26]; and the execution step deals with the action-side of the selected production. In other words, patterns in the buffers hold representations of the external or internal world creating a context to which productions react. One of the productions is selected and fired. This is followed by an update on the buffers creating a new context for the next cycle[27].

Although the decision cycle is clearly a serial process, parallel processes are possible at the same time in different modules given the modular nature of ACT-R.

> The architecture assumes a mixture of parallel and serial processing. Within each module, there is a great deal of parallelism. For instance, the visual system is simultaneously processing the whole visual field, and the declarative system is executing a parallel search through many memories in response to a retrieval request. Also, the processes within different modules can go on in parallel and asynchronously. However, there are also two levels of serial bottlenecks[28] in the system. First, the content of any buffer is limited to a single declarative unit of knowledge, called a *chunk* in ACT-R. Thus, only a single memory can be retrieved at a time or only a single object can be encoded from the visual field. Second, only a single

[25]The version described in this chapter is ACT-R version 4.0. However, because ACT-R is undergoing many changes, some elements of version 5.0 are incorporated as well and are applicable to the design of RBot in chapter 5.

[26]As defined by Anderson and Lebiere (1998, p. 58): "On the basis of the information in the goal, ACT-R must commit to which production to attempt to fire. Because many productions may potentially fire, deciding which production to fire is referred to as conflict resolution."

[27]ACT-R is comparable to a finite state machine that contains a repeating cycle of recognise, decide and act.

[28]See Taatgen and Anderson (2006) and Pashler (1998, 1994) for a discussion on the serial bottleneck.

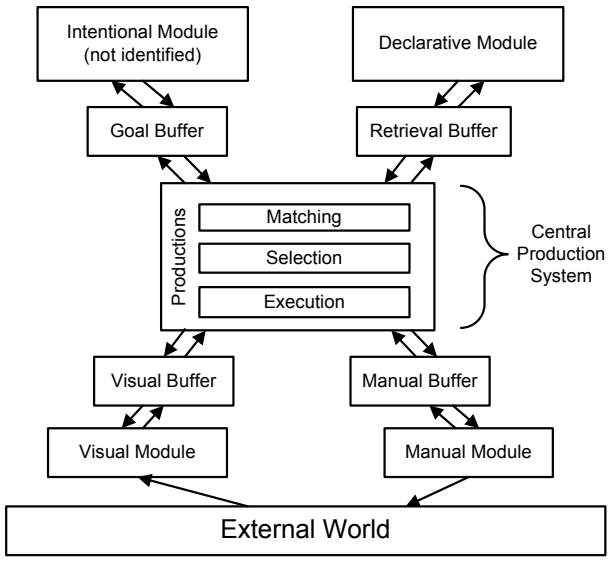


Figure 4.4: The organisation of information in ACT-R 5.0. Information in the buffers associated with modules is responded to and changed by production rules (adapted from Anderson et al., 2004).

> production is selected at each cycle to fire. (Anderson et al., 2004, p. 1038)

ACT-R has evolved and succeeded in integrating mental modules for different cognitive functions into a single theory that exists out of these modules to produce coherent cognition. The functions of the modules—perception/motor, goal buffer/module, declarative and procedural module—are elaborated in the following sections.

### 4.4.1 Perception and Motor module

ACT-R focuses more on higher level cognition than perception or action (Anderson et al., 2004). However, the influence of perception and action, such as perceiving new facts, or learning and acting on objects in the physical world, needs to be considered as having a significant impact on the functioning of the architecture. As previously discussed, situated and embodied cognition have shown that cognition is strongly determined by perceptual and motor processes. Where research concerns social and organisational behaviour as in this dissertation, there is a need for an actor's architecture to require properties enabling interaction and coordination with other actors.

The interaction with the environment was modelled in a specialised version of ACT-R—ACT-R P/M (Perceptual Motor) (Byrne & Anderson, 1998)—that

consists of a separate set of perception and action modules based on the EPIC[29] architecture (Kieras & Meyer, 1997). This extension enriched ACT-R with a "system for manual motor control, parallel operation of system components, and timing parameter for speech and audition" (Byrne & Anderson, 1998, p. 172). The EPIC model has proven itself to be so powerful that, nowadays, it has been adopted in all well known cognitive architectures (Taatgen & Anderson, 2006).

Although the addition of the capabilities of EPIC allows ACT-R to interact with its environment resulting in a stronger and a more comprehensive perception and motor system, the focus of ACT-R is mainly on Human-Computer Interaction and not on interaction with other ACT-R equipped actors. However, despite the lack of attention to the development of task environments for multiple actors in a Multi-ACT-R System, we have confidence that ACT-R is capable of functioning in a multi-actor environment.

### 4.4.2 Goal module

ACT-R's goal module has the purpose of keeping track of the intention of the actor to ensure that the goal is executed in a way that the behaviour of the actor serves the original intention of the goal, i.e. the goal module controls the flow of the task to complete the goal. Originally, in ACT-R 4.0, the goal module was equipped with a LIFO (Last In First Out) goal buffer (stack)—a traditional component in a production system—but was later abandoned, because it produced inconsistent predictions with human data. It was then replaced in ACT-R 5.0 by a "Goal State Buffer" that holds the current control state (Taatgen, 2005). There is still an ongoing debate in the ACT-R community about the ACT-R hypothesis of the single goal structure:

> Many distinct goal modules may manage different aspects of internal state... There is no reason why the different parts of the information attributed to the goal cannot be stored in different locations nor why this information might not be distributed across multiple regions. (Anderson et al., 2004, p. 1042)

In this chapter we adopt the goal stack from ACT-R version 4.0, with the optional (implementation) freedom to restrict the capacity of the buffer to a certain limit[30] and not lose compatibility with ACT-R version 5.0 that has the standard limit of one goal chunk. The goal module is composed of a LIFO buffer (stack) that is initially empty. As ACT-R runs, it pushes a main goal on the stack, which triggers procedures to either change the goal, push a sub-goal or, when the goal is solved, pop (remove) the main or sub-goal. ACT-R's goal module operates as a finite-state machine. This means that ACT-R jumps from one state to the next and when the main-goal is popped (removed), the solution is stored in declarative memory with the help of productions and finishes.

[29] Executive Process Interactive Control

[30] For example in the case of the *Minimal Control Principle*—"the model that is least susceptible to brittleness (cf. Holland, 1986) is the model with the fewest possible values in the Goal State Buffer" (Taatgen, 2005)—the goal buffer is limited to a minimal capacity, e.g. one goal chunk.

### 4.4.3 Declarative module

The declarative module serves as a long-term memory for declarative knowledge. The knowledge is stored as so-called *chunks* that encode small independent patterns of information as sets of slots with associated values (Anderson & Lebiere, 1998). Figure 4.5 exemplifies an addition-fact chunk. Every chunk has a type identifier (isa-slot), a *chunktype*, classifying the pattern formed in the slots of the chunk. Such a classification or mapping supports sorting and can speed up the architectural performance of locating chunks in declarative memory.

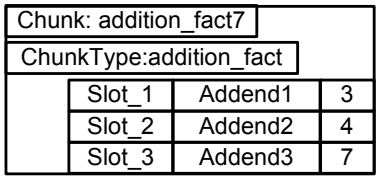


Figure 4.5: Chunk addition-fact 7.

Chunks can be created in two ways: either they are received and encoded from the perceptual module that interacts with the environment, or they are created as a result of a problem that is solved by ACT-R's deliberation process that stores popped goals (goals removed from the goal stack) as chunks in declarative memory.

The declarative memory plays a rather passive role during the resolution of a problem. It is the production memory that determines what is needed from declarative memory in order to solve a problem. The role of declarative memory is to function as a container for chunks that maintain their own activation levels. These activation levels are important for determining the retrieval time of chunks that are stored in the memory. In other words, in the process of storing chunks, activation levels tend not to impose any constraints on time needed for storing chunks. On the other hand, the retrieval of chunks in declarative memory depends strongly on the activation level of chunks; i.e. the activation level of a chunk determines the retrieval time (or the failure of retrieval) of that chunk. The processes of creating and retrieving chunks can have an impact on the activation levels of chunks and thereby on the future retrieval time of chunks in declarative memory; these processes serve as declarative learning processes and are discussed in the following section.

#### 4.4.3.1 Activation, base-level learning and context of chunks

A strong point of ACT-R is its implementation of the functional level by ascribing sub-symbolic properties to the chunks (and procedures) in memory. The sub-symbolic properties of chunks enable the 'strengthening and weakening of information' stored in the memory.

**Activation**

The activation of a chunk (and the utility of procedures, explained in due

course) is one of the sub-symbolic properties that makes an actor to 'forget' and learn. The activation of a chunk is a combination or summation of (1) the base-level activation—reflecting the usefulness of chunks in the past, and (2) the associative activation—reflecting its relevance for the current context (Anderson et al., 2004).

The activation $A_i$ of a chunk in ACT-R is defined as:

$$A_i = B_i + \sum_j W_j S_{ji} + \epsilon \qquad \text{Activation equation}^{31} \quad (4.1)$$

- $B_i$ the base-level activation of the chunk i,
- $W_j$ reflects the attentional weighting of the elements j,
- $S_{ji}$ the strength of an association from source j to chunk i,
- $\sigma$ the (stochastic) noise value, a noise existing out of two components: permanent noise and the noise computed at the time of retrieval based on a logistic distribution[32].

**Base-level activation**

Base-level activation $B_i$ is an estimation of the log odds (of all presentations of a chunk in history) that a chunk will be used and is defined as:

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \beta \qquad \text{Base-level Learning Equation}^{33} \quad (4.2)$$

- $t_j$ represents the time-difference ($t_{now}$ - $t_{presentation}$) that the chunk was presented in memory (created or retrieved),
- $n$ the number of times a chunk is retrieved,
- $d$ the decay rate,
- $\beta$ the initial activation upon creation of the chunk.

The equation implies that the more often a chunk is retrieved from memory (high-frequency), the more its base-level activation rises. On the other hand, the activation level of a chunk that is not retrieved at all can drop below an activation threshold level, whereby it becomes impossible to retrieve the chunk[34].

Figure 4.6 exemplifies the retrieval of a chunk at t = 10 and t = 28. It shows a small (average) increase in activation over the specified period.

As regards the base-level learning equation, Anderson and Lebiere (1998, p. 124) state that "odds of recall... should be power functions of delay, which is the common empirical result known as the Power Law of Forgetting (Rubin & Wenzel, 1996). Although it is less obvious, the Base-Level Learning Equation also predicts the Power Law of Learning (Newell & Rosenbloom, 1981)". Hence, the base-level learning equation enables the actor to learn and prefer chunks that

[31](Anderson & Lebiere, 1998, p. 70)
[32](Anderson & Lebiere, 1998, p. 73)
[33](Anderson & Lebiere, 1998, p. 124)
[34]Actually it could be possible, but only at a very large cost.

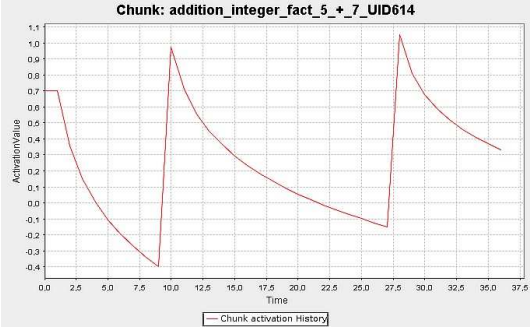


Figure 4.6: Example chunk, accessed at times 10 and 28.

are often needed for solving problems, and to neglect or forget chunks that are (almost) never needed.

The event that causes a change in the activation and presentation of a chunk takes place on three occasions (Anderson & Lebiere, 1998, p. 125)[35]:

1. The creation of a chunk:
   When a chunk is created, the first presentation of the chunk takes place and causes the initial activation. The chunk can be created in three ways:
   **a.** the chunk is added when the actor is pre-configured with knowledge, its birth,
   **b.** the chunk is perceived from the environment,
   **c.** the chunk is created by internal cognition, for example when a goal is solved.
2. The merging of a chunk:
   When a chunk is created and added to memory and the specific chunk already exists, the chunk is "merged" with the existing chunk, resulting in adding a presentation to the existing chunk, avoiding duplication of memory chunks.
3. The retrieval of a chunk *and* strict harvesting of a chunk:
   The production that is involved in the retrieval needs to have a strict reference to the chunk that counts as being presented, i.e. other movements of chunks, without a production intervention, do not count as a presentation.

Base-level learning has been used in many environmental experiments (Anderson & Schooler, 1991) and has been the most successfully and frequently used part of the ACT-R theory (Anderson et al., 2004).

The retrieval of a chunk from memory not only depends on base-level activation, but also on the strengths of association between chunks that (partially) match the demanded chunk.

[35]See also ACT-R tutorial Unit 4 ACT-R version 5.0: Base Level Learning and Accuracy.

**Spreading activations**

Several chunks in memory can be of the same chunktype in which case the contents of the chunk, the slots, become important for matching. For example, the retrieval of a chunk from memory such as an addition-fact chunk with the following slot: slotname == sum and value == 12, many answers are possible (e.g. 1+11 = 12, . . . , 11+1 = 12, but also 1+2 = 12). Hence, the activation is spread over all the (correct and incorrect) facts causing the cognitive system to have more trouble in retrieving the correct chunk from memory. At the same time, latency is increased (see the next section), because activation is reduced by the fan-effect, i.e. the effect of an activation being spread over chunks (Anderson, 1974).

Consider the retrieval of a chunk that has to be matched with the condition side of a procedure (the retrieval part) and assume a request of a chunk with addend1 == 3 and addend2 == 4; figure 4.7 visualises the activation spread.

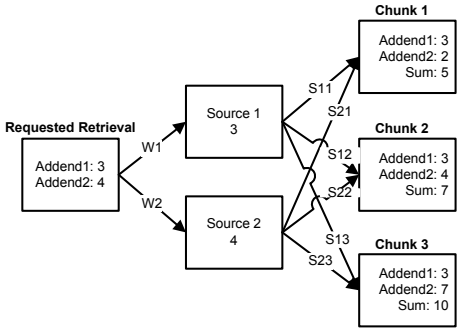


Figure 4.7: Activation spread between goal chunk and three chunks in memory.

The basic equation is:

$$SA_i = \sum_j W_j S_{ji} \qquad \text{Spreading Activation Equation (4.3)}$$

$W_j$ the attention weight which is set to $1/n$ where $n$ is the number of sources and causes an even distribution of strengths,
$S_{ji}$ the strength of the association between source $j$ and chunk $i$,
$d$ the decay rate.

ACT-R has three methods of calculating the activation spread.

1. Normal matching

$$SA_{chunk1} = W_1 * S_{11} + W_2 * S_{21} = 0.5 * 1 + 0.5 * 0 = 0.5$$
$$SA_{chunk2} = W_1 * S_{12} + W_2 * S_{22} = 0.5 * 1 + 0.5 * 1 = 1$$
$$SA_{chunk3} = W_1 * S_{13} + W_2 * S_{23} = 0.5 * 1 + 0.5 * 0 = 0.5$$

2. Fan matching equation

   The general equation used for calculating spreading activation in ACT-R is the fan matching equation. Depending on the spreading of the fan-effect, every chunk gets a certain strength association based on the equation

$$S_{ji} = S - \ln(fan_j) \qquad \text{Activation Spreading Fan Equation}^{36} \quad (4.4)$$

   $S_{ji}$ the strength of the association between source $j$ and chunk $i$,
   $S$ a parameter to be estimated (set with the maximum associative strength parameter),
   $fan_j$ the number of chunks in which $fan_j$ is the linked values of a slot plus one for chunk $j$ being associated with itself.

   The $S_{ji}$ parameter in the previous equation of normal matching is substituted with the activation spreading fan equation that results in the following values for $SA$.

$$SA_{chunk1} = 0.5 * (S - \ln 4) + 0.5 * 0$$
$$SA_{chunk2} = 0.5 * (S - \ln 4) + 0.5 * (S - \ln 2)$$
$$SA_{chunk3} = 0.5 * (S - \ln 4) + 0.5 * 0$$

3. Partial matching

   By not punishing chunks that are closer to the requested retrieval as much as chunks that are further away from the requested retrieval, partial matching looks not only at equality of slots in chunks, but also at the difference of slots in the retrieval, making it possible to assume that 3+2 is closer to the answer than 3+7. The equation of partial matching is:

$$SA_i = \sum W_j S_{ji} - D \qquad \text{Partial Matching Equation}^{37} \quad (4.5)$$

   The $D$ parameter is a mismatch penalty and can be applied to compare numbers, strings and other objects. For instance, for numbers the following equation could be defined (Anderson & Lebiere, 1998, p. 78):

$$D = penalty * (|p_1 - f_1| + |p_2 - f_2|) \quad \text{Mismatch Penalty Equation (4.6)}$$

   In this case, $p_1$ and $p_2$ are the sources (source 1 and source 2) and $f_1$ and $f_2$ are the slots in the targeted chunks (chunk1, chunk2 and chunk3) and $penalty$ is the punishment parameter.
   For the case of figure 4.7, the following can be estimated:

$$SA_{chunk1} = (0.5 * 1 + 0.5 * 0) - penalty * (|3 - 3| + |4 - 2|) = 0.5 - penalty * 2$$
$$SA_{chunk2} = (0.5 * 1 + 0.5 * 1) - penalty * (|3 - 3| + |4 - 4|) = 1$$
$$SA_{chunk3} = (0.5 * 1 + 0.5 * 0) - penalty * (|3 - 3| + |4 - 7|) = 0.5 - penalty * 3$$

[36] See ACT-R tutorial Unit 5 ACT-R version 5.0: Activation and Context.
[37] (Anderson & Lebiere, 1998, pp. 76–80)

When all methods are compared, it becomes apparent that partial matching looks at the associations of chunks at slot-level and is therefore the most accurate estimator. However, the drawback is that the comparison of integers is easier than for instance the comparison of alphanumeric number three with integer 3. The partial matching method is harder to apply. Therefore, the fan equation is recommended in these cases.

Hence, in order to apply partial matching and to calculate the difference of slots of chunks, the cognitive system should provide for the estimation of that difference and to what extent that difference needs to be penalised.

#### 4.4.3.2 Retrieval parameters of declarative chunks

ACT-R not only specifies which chunks are retrieved from memory with the help of an activation level, but is also able to calculate estimations about the probability of retrievals and the time of those retrievals based on this activation level and the threshold. For an architecture to be cognitive plausible, the conditions under which chunks cannot be retrieved anymore and the duration it takes to retrieve a chunk are important.

**Retrieval Probability of a chunk**

ACT-R assumes that "objects [, e.g. chunks,] are never removed from memory ..." (Taatgen, 1999, p. 43) and therefore it does not have the option to remove knowledge. Instead of removing objects, ACT-R applies a threshold value $\tau$ in the retrieval of chunks. When the activation level of a chunk falls below the threshold value $\tau$, that chunk can no longer be retrieved. The threshold value $\tau$ has an impact on the probability of retrieving a chunk from memory. Apart from the threshold, the activation level itself and the noise parameter of the model have an effect on the probability of retrieving a chunk from memory.

This probability is given by the following equation:

$$P_i = \frac{1}{1 + e^{-(A_i - \tau)/s}} \qquad \text{Retrieval Probability Equation}^{38} \ (4.7)$$

- $A$ activation of the chunk,
- $\tau$ threshold value,
- $s$ stochastic variable: $s = \sqrt{3\sigma/\pi}$ where $\sigma^2$ is the combined temporary and permanent variance in the activation levels.

**Probability of retrieving a matching chunk $i$ for a specific procedure $p$**

The matching of a chunk with a procedure depends on the base-level activation and the activation spread. Apart from that, the actual selection is influenced by the noise added to the activation. The probability of retrieving chunk $i$ for procedure $p$ can be approximated with the following equation.

[38] (Anderson & Lebiere, 1998, p. 74)

$$P_{ip} = \frac{e^{M_{ip}/t}}{\sum_j e^{M_{ip}/t}} \qquad \text{Chunk Choice Equation}^{39} \quad (4.8)$$

| | |
|---|---|
| $M_{ip}$ | $M_{ip} = B_i + SA$; matching score between chunk $i$ and procedure $p$, and $SA$ is one of the matching methods; for example partial matching/fan-effect, |
| $j$ | total of matching chunks, |
| $t$ | $\sqrt{2}s$, $s$ equals the stochastic variable in equation 4.7. |

This equation expresses that when chunks with similar matching scores are in the sample and are thereby a suitable candidate for production, the probability for any of these chunks to be retrieved by the production will decrease due to competition between those chunks. Next, even though ACT-R tends to select the chunk with the highest matching score, it gives room for exploration of other chunks due to the influence of noise, i.e. ACT-R is tolerant to the occasional selection of a chunk with a lower score[40].

**Retrieval time and latency**

The total (matching) activation of a chunk (base-level learning, associative strength/partial matching and noise) determines the matching score $M_{ip}$, which individually determines the speed of retrieving a chunk from declarative memory. Chunks with higher activation have the tendency of having lower retrieval times. For instance, a question addressed to the Dutch on the name of the prime minister or president in the Netherlands gives a faster response time than a question addressed to them on the name of the president of for example Zambia. This can be explained by the fact that people are more exposed to information about their local environment than information about other countries. Hence, activation levels of chunks that point to local or familiar information are higher than those for global information.

The retrieval time (or recognition time) of a chunk is determined by the intercept time reflecting encoding and response time, the activation level and a latency scale factor. The equation is:

$$T_{ip} = I + F * e^{-A_{ip}} \qquad \text{Retrieval Time Equation}^{41} \quad (4.9)$$

| | |
|---|---|
| $I$ | intercept time, |
| $A_{ip}$ | activation level or matching score $M_{ip}$, |
| $F$ | latency scale factor. |

[39] (Anderson & Lebiere, 1998, p. 77)

[40] The equation is referred to as the Boltzmann distribution (Ackley, Hinton, & Sejnowski, 1985) or "soft-max" rule because it tends to select the maximum item but not always (Anderson & Lebiere, 1998, p. 65).

[41] This equation is adopted from ACT-R 5.0 (Anderson et al., 2004, p. 1043), a slight change compared to ACT-R 4.0 (Anderson & Lebiere, 1998, p. 80) where production strength is still included. Production strength was included as a parameter with behaviour similar to base-level learning for chunks, i.e. production execution speed was assumed to increase with practise.

The retrieval time equation shows that retrieval time behaves like a negative exponential function. It increases when the activation or matching score diminishes and decreases when the activation or matching score grows.

A procedure can have many retrieval requests to the declarative memory, and the total retrieval time is defined as the sum of all retrievals[42]. The retrieval of several chunks by a production happens one after the other, i.e. ACT-R assumes a serial processing of requests for retrieval of chunks.

The total retrieval time of a series of requests by a procedure is defined as follows:

$$T_p = \sum_i T_{ip} \qquad \text{Total Retrieval Time}^{43} \quad (4.10)$$

$T_p$ total retrieval time of procedure $p$,
$T_{ip}$ retrieval time (equation 4.9) per chunk $i$.

**Retrieval failure**

Whenever a retrieval of a chunk takes place, a retrieval failure may occur when no chunk can be retrieved from memory due to an activation level (or matching score) of the chunk below the threshold value or simply because that chunk does not exist in memory. This results in a retrieval failure of the procedure that attempted to retrieve that chunk. The time needed for the attempt to retrieve that chunk is added to the execution time of the production that causes the retrieval. The time for retrieval failure depends on the threshold value and can be calculated with the following equation:

$$T_f = F * e^{-\tau} \qquad \text{Retrieval Failure Time}^{44} \quad (4.11)$$

$\tau$ threshold value,
$F$ latency scale factor.

In this section, we have explained the retrieval parameters of chunks that have an effect on the performance of retrieving chunks from memory. In the following section, we will elaborate on two types of declarative learning: symbolic and sub-symbolic learning.

#### 4.4.3.3 Declarative chunks and learning: two types of learning

Philosophers have distinguished two sources of knowledge creation by (1) sensing or interacting with the outside world, and (2) cognition or the generation of knowledge by internal processes in the mind. In the same fashion, ACT-R distinguishes two types of creating a chunk: a chunk that is created by the encoding of a "perceptual object, or a chunk created internally by the goal-processing of ACT-R itself" (Taatgen, 1999, p. 43). As an exception, as previously explained,

[42] In ACT-R 5.0, this is restricted in the sense that "... only a single memory can be retrieved at a time or only a single object can be encoded from the visual field" (Anderson et al., 2004, p. 1038).
[43] (Anderson & Lebiere, 1998, p. 80)
[44] (Anderson & Lebiere, 1998, p. 87)

ACT-R does not allow duplicate chunks in memory. It increases the activation of a chunk that already exists in memory (a merger) when trying to create the same chunk.

Hence, learning of chunks can appear at two levels, the symbolic level and the sub-symbolic level:

1. Symbolic level (the creation of a chunk):
   Learning appears by the creation of new chunks: new symbolic structures are created in memory either through perception—the acquisition of new chunks, or cognition—the storing of popped goals and thereby remembering successful reasoning results.

2. Sub-symbolic learning (merging/retrieving of a chunk):

   *Learning based on base-level activation* $B_i$ : the estimation of how likely a chunk is to match a production depends on the frequency and recency of its use. Thus, the more often a chunk is retrieved from memory, the more likely the chunk is re-selected relative to other chunks.

   *Learning based on associative strengths* $S_{ji}$ : depending on the associative strengths and the frequency a chunk is observed, some chunks will be more strongly connected to particular goal requests than others. This type of learning approximates connectionism, in the sense that certain associations or paths from goal to chunks in memory are stronger than others (Anderson & Lebiere, 1998, p. 129).

From studying the previous equations, the notion can be made that procedures interact closely with the chunks in declarative module. All retrievals of declarative chunks are achieved by the processing of productions or the matching of the conditions of productions, i.e. the declarative module is a passive module that depends on the actions caused by the procedural module. In the next section, we explain the function of procedures by outlining the procedural module; a module that recognises patterns and coordinates actions based on these patterns.

### 4.4.4 Procedural module

In ACT-R, procedures occupy a separate place in memory. "The most fundamental distinction in ACT[-R] is the distinction between declarative and procedural knowledge. This distinction goes back to the original ACTE system (Anderson, 1976), and has remained through all the modifications" (Anderson, 1993, p. 18).

Procedures exist out of a condition→action (if→then) pattern. The condition responds to the goal in working memory—the current active goal in the goal stack—and specifies the chunks that have to be retrieved from declarative memory (or other modules) in order to satisfy the condition. When a condition is satisfied in a certain context, the procedure's action-side is triggered. The action-side is able to transform the goal stack by (1) removing the current active goal, (2) adding a new (sub) goal on top of the current goal, or (3) modifying

the current goal. The (possible) remaining statements on the action-side of the procedure allow for interaction with other modules (declarative module, motor module, etc.) in ACT-R, see figure 4.8.

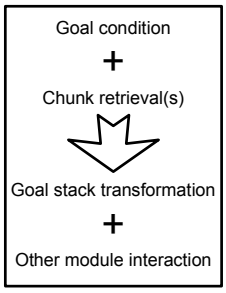


Figure 4.8: Procedure pattern.

An example of a procedure pattern is shown in figure 4.9. The procedure specifies that when the goal is to check vision, it defines a retrieval of a chunk in the perception module. And if there is any information about a car, it reacts on the perception chunk with the slots 'too close' and the driving side variable (e.g. England: =driving equals left, Europe right).

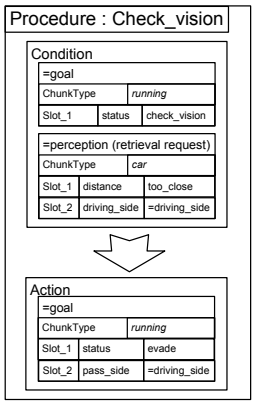


Figure 4.9: Example of a procedure pattern.

On the action side, the procedure defines a modification of the goal chunk, i.e. the slot named 'status' is set to 'evade' and the driving side variable is substituted with left or right (depending on the =driving_side value received from perception). Without these variables, information cannot be transferred from the condition-side to the action-side of the procedure. The consequence of not having variables is that for instance the check_vision procedure needs to be replaced with two procedures in which the variable =driving_side is replaced with the value 'right' and in the other procedure with the value 'left'. Hence, variables create more abstraction in procedures and empower ACT-R with the generalisation of behaviour. In case there is no matching chunk in the perception

module, the procedure is not triggered and another procedure is taken into consideration. As ACT-R has one goal stack and responds only to the goal state, the condition side always contains a single match with the current goal (=goal), while the remaining parts of the condition side can vary in the form of multiple retrieval requests and different module interactions.

ACT-R distinguishes six types of productions possible[45]:

**No goal modification; only action** (No Stack Change, No Goal Modification)
The action-side of the production contains only statements that allow for the execution of actions. However, in order to avoid endless looping—the production should not be selected over and over—the condition-side has to hold a test for state checking. In the case of the car example:

```
Check_vision
  =goal>
     ChunkType       running
     status          check_vision
  =perception>
     ChunkType       car
     distance        too_close
     driving_side    =driving_side
⟹
  =motor>
     ChunkType       action
     pass_side       =driving_side
```

Instead of modifying the goal (=goal), the production immediately changes a slot in a chunk in the motor module (=driving_side). The condition-side in this case tests if there still is a perception chunk with the value 'too close' in it. The production actually functions as a while loop: while there is a goal of check_vision *and* there is a car too close, undertake a motor action.

**Goal Elaboration** (No stack Change, Goal Modification)
This production type is the check_vision procedure in figure 4.9; it modifies the current goal, thereby creating a new (modified) goal state. The new goal state allows other productions to respond to the goal instead of the current one.

**Side Effect** (Push on Stack, No Goal Modification)
A goal is pushed by this type of production that triggers a procedure. For example, the check_vision procedure could be a side effect of driving the car: so now and then, the driver checks if other cars are in its vicinity.

**Return Result** (Push On Stack, Goal Modification)
Some procedures change the current goal, before pushing a new goal on

[45] We apply here the syntax of ACT-R (Anderson & Lebiere, 1998, pp. 41–45).

the stack. For instance, the check_vision procedure is pushed by a procedure X (a Return Result type). When the pushed goal that triggered the check_vision procedure fails, or is popped, procedure X is not selected again, because it changed the goal in the goal stack before pushing a new goal onto the stack.

**Pop Unchanged** (Pop Stack, No Goal Modification)
This type of procedure is applied when a goal needs to be popped and thereby returns to a higher goal in the stack. If it is the last goal in the stack, then it ends the current problem.

**Pop Changed** (Pop Stack, Goal Modification)
This production type is almost equal to the previous, however for storage purposes, it changes the goal that is popped, so that the goal is stored in declarative memory with extra added information.

The execution of those productions is handled by the cognitive engine, explained in section 4.4.5. It checks the goal conditions of all productions, chooses a matching production (the one with the highest utility), performs chunk retrieval(s) if needed, handles goal stack transformations, and possibly other module interactions. At the same time, the cognitive engine in ACT-R has the ability to change the values at the sub-symbolic level of chunks and procedures. First, by implementing a timer in ACT-R, the engine notifies the chunks of a time-change and enables chunks and procedures to construct a time-based history of presentations of the past (base-level learning). Second, based on events like failure or success, the cognitive engine modifies parameters at the sub-symbolic level of procedures that serve as inputs for solving a problem.

In the following section, we explain the symbolic learning that leads to new procedures in procedural memory. In the section 4.4.4.2, we describe the sub-symbolic level of procedures that registers the history of successes and failures.

#### 4.4.4.1 Procedures and symbolic procedural learning

Prior to the start of a simulation, procedures are pre-specified to do the task for which they are designed (Anderson et al., 2004). Aside from probability matching, these procedures indicate the order that the productions ought to assume. Therefore, in the architecture that preceded ACT-R, namely ACT* (Anderson, 1983), there were efforts to discover the basic mechanisms underlying production rule learning. However, these mechanisms were not automatically adopted in ACT-R, because there was generally no evidence to support such learning mechanisms (Anderson & Lebiere, 1998). In the latest versions of ACT-R, versions 5 and 6, production learning mechanisms were sought again and adopted. Therefore, we include here the four mechanisms of ACT* (Anderson & Lebiere, 1998, p. 106):

1. Discrimination. If a rule successfully applied in one situation and did not in another situation, variants of the rule would be

generated by adding condition elements that restricted the productions to the appropriate situations.

2. Generalization. If two rules were similar, a generalized production was produced by either deleting condition elements or replacing constants by variables.
3. Composition. If there was a sequence of rules that applied in a situation, a new single rule could be formed that performed all the actions of the sequence.
4. Proceduralization. If a rule retrieved some information from declarative memory to instantiate an action to be performed, a variant of the rule could be created that eliminated the retrieval and just performed the instantiated action.

The basic motivation for these learning procedures was the observation that people became more tuned in their application of knowledge (discrimination), generalized their knowledge (generalization), came to skip steps in procedures (composition), and eliminated retrieval of declarative knowledge (proceduralization).

In the latest versions of ACT-R[46], *production compilation* (Taatgen, 1999; Anderson et al., 2004) has been introduced as a combination of "two mechanisms from ACT*, proceduralization and composition of two rules into a single mechanism" (Taatgen & Lee, 2003, p. 5). We explain procedure compilation with the help of an example from Taatgen and Lee. Consider the following three production rules to find the sum of three numbers:

| | | |
|---|---|---|
| Rule 1: | IF | the goal is to add three numbers |
| | THEN | send a retrieval request to declarative memory for the sum of the first two numbers |
| Rule 2: | IF | the goal is to add three numbers |
| | AND | the sum of the first two numbers is retrieved |
| | THEN | send a retrieval request to declarative memory for the sum that just been retrieved and the third number |
| Rule 3: | IF | the goal is to add three numbers |
| | AND | the sum of the first two and the third number is retrieved |
| | THEN | the answer is the retrieved sum |

The production or procedure compilation is described as creating faster execution procedures through the incorporation of retrieved past facts as hard facts

[46]At the time of implementing RBot, production compilation was (and maybe still is) in a phase of testing, but because of its importance given in the latest versions and because production compilation refers to skill acquisition (habit of actions), we have included it as part of ACT-R and the theoretical description of the cognitive architecture.

For more discussion and application in cognitive models, we refer to Anderson (1982), Taatgen (1999, 2005) and Taatgen and Lee (2003).

in procedures (proceduralization) in combination with the merging of procedures into one procedure (composition). In the case of proceduralization, retrievals from declarative memory are not necessary anymore and the execution time of the procedure decreases. Composition is a merging process that involves creating a new procedure based on the condition side of the first procedure and adding the action side of the second procedure. For example, when a number of facts need to be retrieved, e.g. the numbers 1, 2 and 3, then there is the possibility of creating two new productions: a new rule 1—a combination of old rule 1 and old rule 2, and a new rule 2—a combination of the old rules 2 and 3. These result in the following new rules:

| | | |
|---|---|---|
| New Rule 1: | IF | the goal is to add 1, 2<br>and a third number |
| | THEN | send a retrieval request<br>to declarative memory for<br>the sum of 3 and the third number |
| New Rule 2: | IF | the goal is to add three numbers<br>and the third number is 3 |
| | AND | the sum of the first two numbers is<br>retrieved and is equal to 3 |
| | THEN | the answer is 6 |

This rule can be further specialised into a simplified single rule that combines all three rules:

| | | |
|---|---|---|
| Combined Rule | IF | the goal is to add 1,2 and 3 |
| | THEN | the answer is 6 |

The reduction of retrievals of knowledge from the declarative memory and the reduction of rules into specialised rules occurs when similar situations occur often and the cognitive system is triggered to be more efficient in the use of resources to solve problems, i.e. reoccurring situations promote the acquisition of skills that eventually lead to habits of action. Finally, such a process will reduce problem solving time and transform a novice into an expert.

#### 4.4.4.2 Procedures and sub-symbolic procedural learning

While symbolic learning is about creating new procedures, sub-symbolic learning is about the change of the procedure itself over time. ACT-R administers parameters for credit management of the procedures involved. In a way similar to base-level learning, the production strength equation[47] in ACT-R 4.0 is defined as the decrease of execution time of a production after frequent use. However, in versions 5.0 and 6.0, this parameter is removed. A reason for this is given by Taatgen (1999, p. 217): "...at the start of an experiment, most task-specific knowledge is still declarative. This declarative knowledge is only gradually

[47]Production strength (Anderson & Lebiere, 1998, p. 132): $S_p = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \beta$

compiled into production rules, providing the speed-up normally explained by strength learning".

The main difference between the selection of productions and declarative chunks is the mechanism behind the selection process. The selection (retrieval) of declarative chunks is based on activation levels, while the selection of productions is based on utility calculations.

**Utility**

In ACT-R, every procedure has a utility. Utility is used in order to make selection possible (conflict resolution) in cases where more than one procedure matches the current goal. ACT-R sorts procedures based on utility and selects the procedure with the highest utility. This process is called conflict resolution and will be explained in section 4.4.5 as part of the cognitive engine. The utility (or expected gain) depends on the probability of success and the associated cost of selecting a particular procedure.

The utility of a procedure is defined as:

$$U = P * G - C + \sigma \qquad \text{Utility Equation}^{48} \ (4.12)$$

- $P$ aggregate learning probability, depending on successes and failures of the procedures (explained later),
- $G$ goal value, the amount of effort (in time) that the actor is in willing to spend in order to achieve the goal, e.g. 10 seconds,
  C estimate of the cost (also explained later),
- $\sigma$ stochastic noise variable.

The noise $\sigma$ is added to the utility to create non-deterministic behaviour. Otherwise, with deterministic behaviour, the selection process of ACT-R would give no room for other productions to be executed (fired), i.e. those whose condition side matches the goal as well.

$G$ or goal-value is the importance ACT-R attaches to solving a particular goal. There is the possibility to assign every procedure a different goal-value that influences the ACT-R engine in selecting a procedure to solve a particular goal. In most ACT-R models, the goal-value is kept equal among all procedures. An exception is the goal stack in ACT-R 4.0; the goal stack adjusts the goal-value to lower values when the goal is a subgoal of a higher goal (explained later).

The learning probability $P$ depends on two sub-probabilities: $q$, the probability of the procedure working successfully, and $r$, the probability of achieving the goal upon the successful functioning of a procedure. $P$ is defined as:

$$P = q * r \qquad \text{Probability Goal Equation}^{49} \ (4.13)$$

- $q$ probability of the procedure working successfully,
- $r$ probability of achieving the goal if the procedure works successfully.

[48](Anderson & Lebiere, 1998, p. 77); (Anderson et al., 2004, p. 1044)
[49](Anderson & Lebiere, 1998, p. 62)

Similar to probability $P$, the cost of a goal is divided into two parameters:

$$C = a + b \qquad \text{Cost of Goal Equation}^{50} \ (4.14)$$

$a$ effort in time of executing the procedure,
$b$ the estimate of the amount of time a procedure takes before the goal is achieved.

Cost $C$ reflects the amount of time necessary for a procedure to complete its task. Cost $a$ is the amount of time that is used to execute the procedure (retrieval cost and production execution) and all subgoals (and sub-procedures) that are triggered by the procedure until the goal that has triggered the original procedure is changed. Hence, cost $a$ reflects the cost of search-paths in memory.

Cost $b$ is the estimated amount of time the procedure takes to achieve the completion of the goal and to solve a (sub) problem. Such costs can create problems, because particularly in tasks that take a long time, e.g. two weeks, the effort invested into accomplishing the goal can be large. Upon the resolution of a problem, the utility of a procedure can drop so dramatically that its re-selection is no longer justified. Future investigations of this problem can provide solutions for time-consuming tasks by creating a dependency between the goal-value (or cost) and the type of task, i.e. the goal-value could be based on the motivation for and expectancy (and passed experience) of a certain task to which it is related.

The $q$, $r$, $a$ and $b$ parameters are affected by experiences over time and allow the cognitive system to operate at all times to optimise the adaptation of the behaviour of the organism (Anderson, 1990, p. 28). We initially explain how successes and failures affect learning. We then discuss the influence of efforts on learning.

**Probability $q$ and $r$**

Probability $q$ and $r$ describe two types of success-ratio of production usage. They reflect the history of successes and failures of a production. $q$ is the success-ratio of executing the procedure, including all retrievals and the execution of, if present, any subgoals. $r$ is the success-ratio calculated over the achievement of completing a goal, after all of the subgoals are solved and the current task and its related goal are met and popped (explained in 4.4.5). $q$ and $r$ are defined as:

$$q, r = \frac{Successes}{Successes + Failures} = \frac{S}{S + F} \qquad \text{Probability Learning Equation}^{51} \ (4.15)$$

$S$ experienced successes,
$F$ experienced failures.

In a simulation setup, the modeller sometimes brings in knowledge in the form of prior past successes and failures. These are combined with current successes and failures, experienced upon the start of a simulation. To make this

[50](Anderson & Lebiere, 1998, p. 62)
[51](Anderson & Lebiere, 1998, p. 135)

distinction clear, the equation is specified as follows:

$$q, r = \frac{\alpha + m}{\alpha + \beta + m + n} \qquad \text{Probability Learning Equation}^{52} \ (4.16)$$

- $\alpha$ prior successes (default 1),
- $\beta$ prior failures (default 0),
- $m$ experienced successes,
- $n$ experienced failures.

**Cost** $a$**,** $b$

As with the success-ratio, costs are also divided into two types. Cost $a$ reflects the cost that is expected to occur when executing a procedure, which includes the rule's retrieval time(s), plus the procedural execution time and additional procedure action time where the action time includes a fulfilling of a subgoal (elaborated in 4.4.5).

Cost $b$ starts the moment cost $a$ stops. Cost $b$ reflects the cost remaining after the procedure is executed until the goal is achieved successfully or is dropped owing to a failure (see also figure 4.11). Cost $a$ and $b$ are defined as:

$$a, b = \frac{Efforts}{Successes + Failures} = \frac{E}{S + F} \qquad \text{Cost Learning Equation}^{53} \ (4.17)$$

- $E$ experienced efforts,
- $S$ experienced successes,
- $F$ experienced failures.

The differences, compared to the Probability Learning Equation (4.15), are that successes are replaced with the effort of executing and fulfilling a task and those efforts are a summation of all the efforts experienced by a procedure, no matter if the experience is a failure or a success. Similar to the Probability Learning Equation (4.16), we can specify the prior successes and failures, a factor $z$ that represents prior influence of efforts, the summation of efforts experienced during the simulation, and the experienced successes and failures of a procedure during the simulation. This can be expressed with the following equation:

[52](Anderson & Lebiere, 1998, p. 135)
[53](Anderson & Lebiere, 1998, p. 135)

$$a, b = \frac{z + \sum_{i}^{m+n} effort_i}{\alpha + \beta + m + n} \qquad \text{Cost Learning Equation 2}^{54} \quad (4.18)$$

| | |
|---|---|
| $z$ | total prior effort (for $a$ default 0.05, for $b$ default 1), |
| $\sum effort_i$ | summation of all experienced efforts (effort typically measured in time), |
| $\alpha$ | prior successes (default 1), |
| $\beta$ | prior failures (default 0), |
| $m$ | experienced successes, |
| $n$ | experienced failures. |

The similarities between the success-ratio crediting and the cost crediting is that success-ratio $q$ and cost $a$ are credited at the same time. The success-ratio $r$ and cost $b$ are also credited at the same time. The details about the credit assignment are explained in section 4.4.5. In addition to the calculation of successes, failures and efforts, ACT-R applies discounting: events and efforts that happened in the past have less of an impact than present events and efforts.

**Event & Effort Discounting**

In the previous equations for calculating the success-ratio and the cost of efforts, no time component was included. Under such a condition, events and efforts in the past are equally weighted as the ones experienced at present time. However, people are often more aware of the impact of present experiences (of the same procedure) than experiences encountered in the past. ACT-R has a mechanism that takes care of discounting past experiences by implementing an exponentially decaying function that is similar to the base-level learning equation. The equation for discounting successes and failures is:

$$Successes, Failures = \sum_{j=1}^{m,n} t_j^{-d} \qquad \text{Success Discounting Equation}^{55} \quad (4.19)$$

| | |
|---|---|
| $m$ | number of successes, |
| $n$ | number of failures, |
| $t_j$ | time difference, now - occurrence_time of the success or failure, |
| $d$ | decay rate (default: 0.5). |

This equation can be substituted in equations (4.15), (4.16), (4.17) and (4.18). This substitution results in a set of new functions for calculating learning probability and cost that depend on the decaying of successes and failures over time.

The effect of the equation on the selection mechanism is that there are more exploration possibilities caused by the decay of past successes and failures. For example, it is possible to give different decay rates for successes and failures, i.e. when the decay rate of failures is lower than that of successes, ACT-R tends to forget negative experiences more slowly than positive experiences.

[54](Anderson & Lebiere, 1998, p. 136)

[55](Anderson & Lebiere, 1998, p. 141) & (Anderson & Lebiere, 1998, p. 265)

The same kind of discounting can be applied to efforts discounting:

$$Efforts_{a,b} = \sum_{j=1}^{m,n} t_j^{-d} Effort_j \qquad \text{Effort Discounting Equation}^{56} \ (4.20)$$

| | |
|---|---|
| $a$ | cost $a$, |
| $b$ | cost $b$, |
| $m$ | iterator representing efforts belonging to cost $a$, |
| $n$ | iterator representing efforts belonging to cost $b$, |
| $Effort_j$ | Effort at iteration $j$ (in time units), |
| $t_j$ | time difference, now - occurrence_time of the success or failure, |
| $d$ | decay rate (default: 0.5). |

The equation expresses that efforts associated with cost $a$ and $b$ are discounted over time resulting in past efforts to be forgotten over time. The efforts in equations (4.17) and (4.18) can be substituted with the equation stated in equation (4.20). This creates new equations that include the decay of past efforts.

Discounting creates the possibility that procedures—that are not used for a long time, because they are credited with high cost—still can be selected in the future. Such an opportunity is possible when other procedures create errors in an environment that demands other capabilities of the actor. When these capabilities are in the procedures that are outranked owing to their high cost, they have a higher chance of being selected in the new situation when efforts have decreased over time. Otherwise, these procedures may not be selected again, because the punishment of a high effort never decreases over time.

The following section explains the cognitive engine that controls the matching of procedures, the selection by means of *conflict resolution* and the execution and credit assignment at the sub-symbolic level.

### 4.4.5 The cognitive engine

The cognitive engine defines a clever mechanism by connecting the procedural module to patterns in other modules of ACT-R. The mechanism consists of cycles and each cycle consists of a certain number of steps. In other words, the engine of ACT-R is constructed as a finite state machine defining a loop, which has the following flow:

1. The condition parts of all procedures are matched to the pattern of the current goal. This results in a list of potential candidates from which one is selected for execution in the end. However, if no candidates are found, the current goal is removed and the next goal in the stack becomes the current goal. The engine returns to step 1 until there are no goals left anymore, in which case, the engine halts.

[56](Anderson & Lebiere, 1998, p. 141)

2. The conflict resolution occurs when more than one procedure matches the goal. In that case, the procedure with the highest utility is selected.

3. The selected procedure is executed (fired). However, when there are any remaining conditions left—such as retrievals from other modules (e.g. declarative memory or perception)—then these are processed first. When these retrievals result in retrieval errors, the production fails prior to processing the action side. The engine then goes back to step 2.

   If the engine succeeds in processing the condition-side without any failures, then the action-side of the procedure will be handled. The action-side contains actions that can change the contents of the goal stack and/or interact with other modules.

4. When execution succeeds, or failure takes place, the engine starts the accounting of values to the relevant sub-symbolic parameters of the procedures and chunks.

5. There is a return to state 1 upon the existence of a goal in focus (current goal), otherwise the engine exits. (halt)

One side condition is necessary for ACT-R to avoid endless looping of the engine: the goal module must always change[57] to avoid loops in a single cycle. However, endless cycles can occur when the next procedure puts the goal module in its previous state. If there is no escape, ACT-R will go into an endless loop[58].

The ACT-R engine and the interaction with the memory modules is shown in figure 4.10.

The ACT-R engine reacts based on the contents of the current goal. The current goal is matched with all the procedures. The procedures that match the current goal are candidates for firing. When there is more than one procedure that is a candidate for execution, conflict resolution takes place. Conflict resolution is a process that compares the utilities of all candidates and selects the procedure with the highest utility.

After a procedure has been selected, the necessary retrieval requests to memory are made. When no retrieval failures are reported, the action side of the procedure is executed. Change such as popping, modifying or pushing a goal in the goal stack occurs on the action side of the procedure.

As mentioned before, ACT-R states that popping (removal) of a goal occurs with placing the popped goal from the goal stack into declarative memory. In this way, new chunks are learnt based on cognitive activity. Besides the creation of declarative chunks, new productions are created with the help of production compilation, which is an autonomous process inside the engine that creates new procedures out of old procedures and/or chunks in memory.

The inner working of the ACT-R engine, i.e. the five steps of the ACT-R engine, will be explained in the following sections.

[57] There is one exception (see section 4.4.4): the procedure type *no stack change, no goal modification.*

[58] This strongly depends on the developer who is constructing the procedures. There is not yet an escape mechanism that prevents an ACT-R to avoid entering a vicious loop.

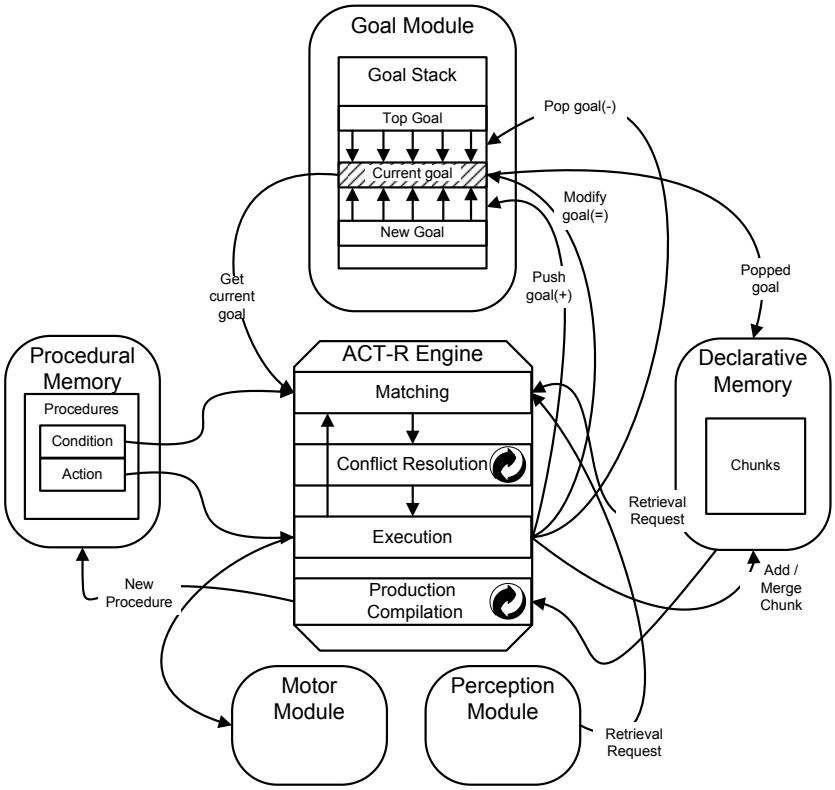


Figure 4.10: ACT-R engine and interaction.

#### 4.4.5.1 Matching (Step 1)

The ACT-R engine compares the goal chunk with the =goal statement in the procedure condition. It checks all the procedures and sees if a match is there. When a match is found, it is added to a list of procedures that have the potential to fire. In ACT-R, the matching process does not involve a check for the success or failure of retrieval requests. This is handled in the execution step.

#### 4.4.5.2 Conflict Resolution (Step 2)

After matching all procedures, a set of matching procedures is obtained. ACT-R, in contrast to SOAR, only fires one procedure at a time. The matching procedures are seen as being in conflict with each other and a process, *conflict resolu-*

*tion*, has to make clear which procedure will eventually fire. Figure 4.11[59] shows the selection of a procedure (P4) based on conflict resolution.

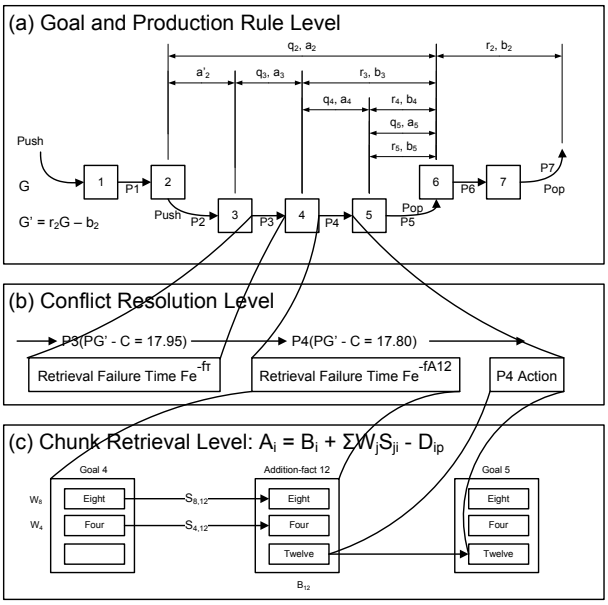


Figure 4.11: Conflict resolution / credit assignment (Anderson & Lebiere, 1998, p. 59).

As regards conflict resolution, the procedures that are important in figure 4.11 are procedures P3 and P4. Procedures P3 and P4 are in the potential procedure list, of which procedure P3 is the winner, because its score of 17,95 is higher than 17,8. However, after trying the retrieval request on the conditional side of procedure P3, the procedure fails and procedure P4 is the next in rank to be selected. This procedure has no retrieval errors and can be executed (step 3) without any problems[60].

[59]The process of solving a goal is as follows: the goal G is pushed on an empty goal stack; this leads to state [1]. Production P1 is triggered by this state and changes state [1] into state [2]. Production P2 pushes a new goal on top of the goal stack, and this leads to state [3]. The goal-value [*G*] is reduced to [*G'*] when a goal is pushed; 'deeper' goals are less likely to bring success in solving the problem.
State [3] triggers P3, which tries to retrieve a fact from declarative memory. However, as can be seen at (b), the conflict resolution level, P3 results in a retrieval error. In that case, the next procedure, P4 from among possible candidates, is selected for firing (thus, state [3] equals state [4]). At the chunk retrieval level (c), P4 succeeds in finding a chunk (addition-fact12) and changes state [4] in state [5] by filling in an empty slot with 'Twelve'. Production P5 pops the goal and brings state [5] to state [6]; back to goal level G. Production P6 changes state [6] to state [7] followed by production P7 that pops the goal and empties the goal stack.

[60]In case no procedure can be found at all, the goal (e.g. goal4) will be dropped by ACT-R with a failure.

#### 4.4.5.3 Execution (Step 3)

In step 3, if any statements remain on the condition part of the procedure, they are checked out by trying to retrieve those statements from declarative memory or other defined modules. This means that there is a set of retrieval requests for chunks in memory that has to be executed. If that step fails (procedure P3), the ACT-R engine goes back one step and selects, in this case, procedure P4 for execution. The condition side of procedure P4 appears to have no problems, so the action side of P4 is executed and state [4] is transformed to state [5] with the third slot of the goal (Goal 5) filled with number 'Twelve'.

#### 4.4.5.4 Credit assignment (Step 4)

Credit assignment is the process of assigning values at the sub-symbolic level. Examples of credit assignments are (1) values of goal levels ($G$ and $G'$), (2) the sub-symbolic values $q$ and $a$: the probability and time for executing a procedure, (3) the procedure values $r$ and $b$—after execution, the probability of achieving the goal and the estimate of the amount of time a procedure takes before the goal is popped, and (4) notifying chunks when they are retrieved for the purpose of base-level learning.

**Goal level credit assignment**

ACT-R makes a distinction in goal values at different levels of goal stack execution. The stack in figure 4.11 appears to be two levels deep. ACT-R calculates the goal-value at level 2 ($G'$) by discounting the goal-value at level 1. The goal-value is calculated based on the expected future effort and success of achieving the subgoal. The equation for estimating the goal-value of a goal at a lower level is.

$$G' = G_{level_{x+1}} = r * G_{level_x} - b \qquad \text{Goal Discounting}^{61} \quad (4.21)$$

| | |
|---|---|
| $G_{level_x}$ | goal at level/depth $x$ of the stack, |
| $r$ | probability of achieving the goal if the procedure works successfully, |
| $b$ | the estimate of the amount of time a procedure takes before the goal is achieved. |

> $G'$ is *discounted* from $G$ to reflect uncertainty ($r$) and cost ($b$) in achieving the goal even after the subgoal is achieved. Thus, $G'$ reflects the maximum amount that should be expended in achieving the subgoal and this is the value attributed to this subgoal. Thus, the value assigned to a subgoal depends on the context in which it occurs. (Anderson & Lebiere, 1998, p. 63)

ACT-R shows that the lower a goal level is, the higher the probability that the problem is unsolved at that level. We argue that this is cognitive plausible, because human beings are boundedly rational and are (normally) not able to

[61] (Anderson & Lebiere, 1998, p. 63)

memorise goal structures with many subgoals. We state that the goal-value can reflect the capability and motivation for solving goals.

**Credit assignment of $q$ and $a$**
There are two possibilities following the execution of the procedure:

1. No subgoal is pushed and the ACT-R engine immediately can assign credits to $q$ and $a$ (figure 4.11, $q_3$ and $a_3$). When a procedure (P3) causes retrieval errors and fails, the number of failures ($F$) for $q$ and cost $a$ is increased with 1 and the associated retrieval failure time is estimated as $Fe^{-f\tau}$ (equation (4.11)). However, when the retrieval requests are successful (P4), both $q$ and cost $a$ are credited success and the retrieval time becomes $Fe^{-fA}$.

2. If a subgoal is pushed by a procedure (figure 4.11, P2), credit assignment (a failure or success) for this procedure takes place the moment this subgoal is popped (P5 pops the goal). Cost $a$ of P2 is based on the successes or failures and the summation (total time) of all $efforts(a)$ of the procedures that were involved in completing the subgoal.
$effort(a_2) = effort(a'_2) + effort(a_3) + effort(a_4) + effort(a_5))$.

**Credit assignment of $r$ and $b$**
Credit assignment to $r$ and $b$ takes place in the following cases:

1. The procedure pops the current goal (figure 4.11, P5 & P7). When the procedure pops the current goal, ACT-R assigns $r$ and $b$ a success or effort similar to that assigned to $q$ and $a$ of that procedure, e.g.
$< (S)r_5, (S, effort)b_5 > = < (S)q_5, (S, effort)a_5 >$

2. The procedure is followed by another procedure at the same level. For instance, procedure P3 is followed by P4 at the same level. In this case $r_3$ and $b_3$ cannot be assigned immediately to P3, only when the goal at the level of P3 is popped. In figure 4.11, procedure P5 pops the goal at the level of P3 and at that point, credits to $r$ and $b$ to all procedures (P3, P4 and P5) at that level can be assigned.

   In the case of P3, there are two possibilities: The successes of $r_3$ and $b_3$ depend on P5; if P5 succeeds in dropping the goal, then P3 will be credited with successes: $(S)r_3 = (S)b_3 = (S)q_5 = (S)r_5 = (S)a_5 = (S)b_5$

   A special case occurs when P5 fails and no other procedure matches, then the goal fails and $q_5$, $r_5$, $a_5$, $b_5$, $r_3$ and $b_3$ are assigned a failure:
   $(F)r_3 = (F)b_3 = (F)q_5 = (F)r_5 = (F)a_5 = (F)b_5$

   In both failure or success cases, the effort of $b_3$ depends on the effort of procedure P4 (cost $a_4$) and the effort of procedure P5 (cost $a_5$)
   See figure 4.11: $(effort)b_3 = (effort)a_4 + (effort)a_5$

**Credit assignment of procedure execution**
The only remaining effort that needs to be added to every procedure is the effort taken by the cognitive engine to execute the procedure, i.e. during every execution of a procedure, a standard execution effort of 50ms is added to the efforts of the fired procedure ($effort(a'_2)$, under the assumption that procedure P2 has no retrieval of chunks).

**Assigning of presentations of retrieval or merger to chunks (base-level learning)**
In the execution phase, the retrievals have to be registered at the sub-symbolic level of chunks as well as the merging of chunks when goals are popped or when new chunks from perception are encoded and merged.

#### 4.4.5.5 Check end-state (Step 5)

In Step 5, which can also be seen as Step 0, the ACT-R engine inspects the goal stack to see if there are any goals left to fulfil: if a goal is present, the engine uses that goal for the matching of procedures and moves to step 1. If there are no more goals, the engine stops. (on halt)

The description of the core components given thus far is sufficient to start the implementation of simulation software, i.e. see RBot in the next chapter, which is dedicated to the study of organisational behaviour of a collection of cognitive plausible actors.

Prior to the design and implementation of RBot, we evaluate ACT-R and explore what is missing and what adaptations are necessary for it to fulfil the requirements for the construction of a Multi-Agent System that is able to simulate social and organisational behaviour.

## 4.5 Evaluation of ACT-R

The evaluation of ACT-R will be based on the three research questions mentioned in chapter 1. The research questions address what is necessary for the implementation of a cognitive agent-based social simulation tool. In order to develop such a tool with the help of ACT-R, we first evaluate ACT-R and suggest directions, changes or extensions that are needed for ACT-R to function as a cognitive plausible (4.5.3) and socially situated (4.5.2) Multi-Agent System (4.5.1).

### 4.5.1 Multi-Agent System

The first question (1.1) addressed what type of model was suitable for explaining interactive social behaviour. We have argued that a Multi-Agent System (MAS) suffices as a model and a methodology to explain interactive social behaviour. Although ACT-R is not a Multi-Agent System and is primarily focused on the individual and its interaction with the environment (and not on interaction with other actors), there have been some developments of ACT-R agents interacting in small teams within a spatial domain and real-time environments (Best

& Lebiere, 2003, 2006). These developments have "demonstrated that the same framework for action and perception at the cognitive level used in the ACT-R agents [...] can be used to control behavior in both virtual and robotic domains" (Best & Lebiere, 2006, p. 216). This shows that ACT-R is capable of interacting and communicating in a virtual as well as a physical environment and can serve as a plug-in for other environments.

A sustainable development of a Multi-ACT-R model is however still lacking owing most likely to the cognitive psychological roots of the ACT-R community. On the other hand, Multi-Agent Systems are part of Distributed Artificial Intelligence, which is a community that is interested in problems solved by coordinating interactions between different actors distributed over several threads, on the same computer or distributed over several computers in a network. In order to bridge the gap between MAS and ACT-R, some modifications are necessary to pose ACT-R as a serious candidate for Multi-Agent Systems.

**Adjusting the design of ACT-R: integration in a MAS.**

The first step for ACT-R is (1) the need to have an artificial (simulated) task environment in order to construct various multi-actor environments and interaction scenarios. These interactions can be accomplished by the exchange of signals and signs between the actor, the (physical) environment and other actors, as described by the model of Best and Lebiere (2006).

In the next chapter, we introduce RBot—a cognitive social daughter of ACT-R—and MRS (Multi-RBot System) that implements an artificial task environment. The creation of such an artificial task environment that is suitable for multi-agent interaction suggests a need for a set of ACT-R agents that can be distributed over a network, i.e. a network that consists of heterogeneous computer systems[62]. This requires (2) an ACT-R agent that is portable to different computer platforms, and (3) requires an Agent Communication Language (ACL) that enables ACT-R agents to communicate with each other and if possible with other types of actors.

These are three adjustments we have applied to ACT-R, which have resulted in a new cognitive actor RBot. The first and radical adjustment that fulfils requirement (2) is the choice for implementing ACT-R in the programming language JAVA™. JAVA is an object oriented programming language that allows the execution of programs on a virtual machine that can be integrated in many well known flavours of operating systems[63]. The most important reason for the shift to JAVA is most likely its advancement in conjunction with the Internet, the large user community and the immense library of tools that can be integrated into the development process. The second adjustment we applied to ACT-R and which satisfies requirement (3), is the implementation of message-based communication with the help of speech acts or an Agent Communication

[62]This is a rather strong implication; we want to address the heterogeneity of systems that are connected to the Internet. Although, theoretically, a MAS could run on a single machine, we assume that future applications of ACT-R and RBot will run on distributed computers.

[63]ACT-R is programmed in LISP and also runs on different platforms. However, LISP's popularity is not as widespread as JAVA; in the sense that it is well recognised in the AI community, but not beyond. Therefore, JAVA is a good candidate for introducing ACT-R *and* RBot to other communities.

Language. An ACL such as FIPA (FIPA, 2002) allows actors to exchange information according to a format or code that is known to all agents involved and thereby supports 'open' communication between agents of any kind. The third adjustment is the implementation of a generic artificial task environment; MRS (Multi-RBot System). The MRS is a simple server system that represents a centralised artificial task environment and facilitates physical and communicative interaction between agents, i.e. the server has the purpose of providing an artificial 'physical' task *and* a communication environment that allows actors to perceive objects and other actors, and of supporting the exchange of signals and signs. The next step is the step of social interaction and distributed problem solving with the help of representations of (joint) tasks, social structures and roles in the minds of actors.

### 4.5.2 Social behaviour: social interaction and social constructs

The second research question addresses the requirements that are necessary for an actor to exhibit (stable) social behaviour and that such behaviour requires the creation of shared knowledge, social construction and semiotic resources that include social structures, institutions and habits of action.

We refer back to the levels of description where we distinguish levels varying from the physical or biological to that of social.

In a similar vein, Newell (1990) describes a time scale whereby different bands are identified for human action as shown in table 4.3.

| **Time scale of human action** | | | |
|---|---|---|---|
| Scale (sec) | Time Units | System | World (theory) |
| $10^7$ | months | | |
| $10^6$ | weeks | | **SOCIAL BAND** |
| $10^5$ | days | | |
| $10^4$ | hours | Task | |
| $10^3$ | 10 min | Task | **RATIONAL BAND** |
| $10^2$ | minutes | Task | |
| $10^1$ | 10 sec | Unit Task | |
| $10^0$ | 1 sec | Operations | **COGNITIVE BAND** |
| $10^{-1}$ | 100 ms | Deliberat Act | |
| $10^{-2}$ | 10 ms | Neural circuit | |
| $10^{-3}$ | 1 ms | Neurons | **BIOLOGICAL BAND** |
| $10^{-4}$ | 100 $\mu s$ | Organelle | |

Table 4.3: Time scale of human action (Newell, 1990, p. 122).

> Starting from the bottom, there is the *biological band* of three levels—neurons; organelles, which are a factor of ten down from neurons; and *neural circuits*, a factor of ten up...Above the biological band, there is the *cognitive band.* Here the levels are unfamiliar—I've called them the *deliberative act, cognitive operation, and unit task...*

> Above the cognitive band lies the *rational band*, which is of the order of minutes to hours. All of these levels of the rational band are labelled the same, namely, as *task*. Finally, even higher up, there lies something that might be called the *social band*,... (Newell, 1990, pp. 122–123)

ACT-R, similar to SOAR, focuses on the individual and more specifically at the cognitive band and the rational band[64]. At the cognitive band, productions fire on average at every 100 ms (Anderson & Lebiere, 1998, p. 13) and the rational analysis takes place at the rational band ($10^2$-$10^4$). According to the time scale of human action social behaviour lies at $10^5$-$10^7$ sec (2,8 hr - 116 days). There is a gap between the social band, the rational band and the cognitive band, varying from $10^2$ (rational band) till $10^6$ (cognitive band, i.e. production level) that needs to be bridged, i.e. social behaviour needs to be explained by behaviour of the individual (the cognitive and rational band) and the other way around.

**Adjustment of ACT-R to allow for social interaction**

In order to bridge this gap, we have argued that a semiotic level—see chapter 1 & 3—is necessary between the individual and the social level. We do this by defining the *social construct*[65] as a (social) representation in the mind of the actor, or as documents or artefacts in society, reinforced by their frequent use resulting in habits of action or social norms.

The implementation of a multi-agent task environment allows one to simulate and study social behaviour and interactions between individuals. ACT-R has the capacity for storing chunks or rules that depend on the social context or interactions with others. However, ACT-R is not designed for the purpose of explaining social behaviour. In other words, there are no social properties or mechanisms defined. Therefore, we have created the social construct as a separate unit of (social) knowledge. The social construct enables one to define social situations in which the actor has to obey or is permitted to follow certain rules that are present in a society or culture. The implementation is carried out by creating a special module in which social constructs are entities that influence certain aspects of behaviour. Imagine, for example, that a certain situation does not allow for certain goals or the execution of certain rules.

The mechanism that is adopted for implementing social constructs is the subsumption architecture (Brooks, 1986); a multi-level system in which the upper (normative) level is a social construct that influences behaviour at the lower level(s). The concept of social construct allows us to model social situations, emergent social behaviour and interactions with other actors, which is impossible to accomplish in ACT-R.

[64]The biological band is not the main focus of SOAR, ACT-R *and* this dissertation; for a description, see (Newell, 1990, p. 123).

[65]See chapter 3 for an explanation about the social construct.

### 4.5.3 The rational and functional level

The third research question addresses the kind of actor that can plausibly handle signs, relations and social constructs. We argue that a cognitive architecture like ACT-R is a physical symbol system that is able to handle signs and, as explained in the previous section, can handle and store social constructs.

**Adjustments at the rational level: goal deliberation**

ACT-R is an architecture that uses a rational approach (Anderson, 1990, p. 30) to study human behaviour. It does this by specifying what the goals are of the cognitive system, developing a formal model of the environment to which the system is adapted and making minimal assumptions about computational limitations such as the assumption of short-term memory (Anderson, 1990, p. 29).

The structures above the atomic components (declarative chunks and productions) are goal structures that provide a "chemical structure" (Anderson & Lebiere, 1998, p. 13). In ACT-R 4.0, there was a goal stack that provided such a goal structure, but this was abandoned in ACT-R 5.0, because it did not conform to predictions found in experimental data. We argue that, because there is not an intentional module defined in ACT-R, it is unclear how one can incorporate the "chemical structure" of goals in a plausible way. Although we do not provide experiments that can give directions for cognitive plausible structures, we assume that structures such as tree-like structures and a deliberation module similar to a Beliefs Desires Intentions[66] module or a motivational subsystem (Sun, 2003) is necessary for the maintenance of (social) goals.

Interaction with other actors requires more control over goals, because many collaborations with others actors depend on planning, commitment and exchange of goals. In RBot, discussed in the next chapter, we have implemented a goal list (a simple FIFO buffer) that provides the opportunity to exercise more control over the processing of goals and is able to address issues of planning or dual tasks, that is:

> The goal list can handle more complex or more subtle situations that the goal stack cannot easily handle. For example, in order to schedule several activities at the same time, such as carrying on a conversation while searching for a file in the desk drawer, the priorities of different goals, such as "continuing conversation" and "continuing file search", can change dynamically. As the delay time since a question was asked grows, the goal of "continuing conversation" becomes increasingly more important. On the other hand, when the conversation drags on, the goal of "continuing file search" may become dominant again. This kind of alternation of goals may be difficult for the goal stack to handle. (Sun, 2003, p. 50)

[66] The BDI (practical reasoning) theory is not applied in this dissertation, because it operates mainly at the rational level. However, the BDI theory can give alternatives or suggestions for the implementation of an intentional module (cf. Bratman, 1987, 1990; Rao & Georgeff, 1995; Wooldridge, 2000).

Sun (2003)'s architecture CLARION focuses more on the individual system than on cooperation with other actors. He elaborates on the problems that occur when there is not a separate structure that handles goals. He promotes the implementation of a motivational and meta-cognitive system that deals with the priorities of goals. We encountered similar problems when we assigned multiple goals to the actor. Therefore, we also incorporated a simple goal structure or list in RBot.

**Adjustments at the cognitive level**

The adjustments at the cognitive level are minor adjustments, compared to the adjustments at the higher level. As the next chapter shows, many mechanisms and components defined in ACT-R are not changed. In this manner, we probably maintain the cognitive plausibility of ACT-R that is seen as a necessary outcome of thorough testing over the last 20-30 years. Nonetheless, we will make a few slight adjustments as regards the memory structure, the structure of productions, and issues related to the base-level decay and estimation of efforts.

**Memorymap and memory structure** ACT-R creates a clear distinction in several memory modules that each have a certain functionality. In RBot, we have created the memorymap as a reference container for chunks. Some of the chunks in the memorymap are assigned to be an entrance to the contents of the memorymap. The other chunks in the memorymap are accessible via the entrance chunk, i.e. the other chunks have links with the entrance chunk and can only be reached by the entrance chunk. Because of the fact that a memorymap maintains a collection of references, chunks can be member of several memorymaps at the same time. Secondly, there exist the possibility of adding links or paths (of association) between chunks that are in different memorymaps.

In the next chapter, we will explain more thoroughly about this adjustment of memory management. The adjustment gives more flexibility in creating memory structures compared to ACT-R.

**Structure of productions** ACT-R distinguishes a production and a declarative memory that contain units of knowledge: productions and declarative chunks as containers of knowledge. RBot distinguishes a procedural and declarative memory as well. However, an attempt is made to reduce the units of knowledge to a single unit: the chunk. When we carefully study the syntax of a production in ACT-R (Anderson & Lebiere, 1998, pp. 31–34), we can recognise patterns of (goal) chunks in the production itself. Such a pattern (ibid., p. 34) can be a pattern-chunk (a chunk that contains variables) and these pattern-chunks can be linked together to form a production.

In other words, the production can contain a condition- and an action-chunk that is linked with pattern-chunks—a network of linked chunks—that together form the production. The production is a chunk consisting of chunks (see the next chapter for a detailed explanation).

**Base-level decay** The decay parameter is a problem in the sense that the parameter has to be adjusted depending on the length of the experiment (see Taatgen, 1999).

> A fast decay of 0.5, which is the official recommended value of the parameter, turns out to work best for decay within an experimental session. A slow decay of 0.3 is necessary for experiments in which an hour, a day or a week passes between experimental sessions, else ACT-R will forget all it has learnt. (ibid., p. 217)

Especially in experiments on social interaction that can take longer than a (simulated) week, the decay rate is too high. Alternative solutions should be provided to prevent the loss of social constructs that are formed in interactions occurring in the social band, a time band that can surpass hundred days.

A possible solution can be the adjustment of the decay function. For example, after an interaction or an experimental session has ended, a lower decay parameter becomes active that slows down the decaying process; see Taatgen (1999, p. 217) and Anderson, Fincham, and Douglass (1999, p. 1133). In our experiments, we assume that social constructs are habits of action and are somehow similar to productions, i.e. in our current version of RBot social constructs are not subjected to decay. Therefore we are not confronted with this problem in our experiments.

**Estimation of effort** The cost or effort of reaching a goal also needs to be considered. In ACT-R, the maximum effort that a task takes is between 10 and 15 seconds. However, when a task (and goal) lasts longer, and especially in social interaction and experiments where the task can take a long time before it is solved, the impact of the effort on the cost of the utility to reach a goal is not realistic. For example, when an actor travels from A to B in about 15 minutes, the cost of reaching that goal is high compared to tasks that take a shorter time. The next time the actor tries to solve the goal, it will not select the procedure that responds to the goal. This is not due to the procedure being unsuccessful, but simply the fact that it will cost too much effort to even think about executing the procedure.

The utility function of the procedure is composed of (1) the probability that a goal will succeed and (2) the cost of a successful goal. The problem of cost $b$ we encountered in our models (and especially huge efforts) points to a clear dramatic drop in the utility of the procedure. We solved this problem roughly by limiting cost $b$ to a maximum of 15 seconds. However, other solutions are possible. For instance, another solution could be to estimate the level of cost for achieving the goal in the beginning (a maximum aspiration level) and to compare the realised effort with the aspiration level, which results in a cost function that reflects relative cost. A combination of this solution with a motivational subsystem, which influences the goal-value, could be a good candidate. For instance, if there is a lot of repetition of efforts and the amount of effort invested results in an actor's fatigue, then the motivation may drop. This can have a negative impact on the goal-value of the procedure, encouraging the actor to take a break.

## 4.6 Discussion

In this chapter, we focused on the mind of the individual, i.e. on cognitive science, cognitive architecture, and specifically ACT-R. We adopted ACT-R as a cognitive architecture because it has proven its cognitive plausibility over the last 20-30 years. The analysis of the individual based on levels of description has introduced the notion that an intelligent actor should be a physical symbol system; a system that is a necessary and sufficient condition to exhibit general intelligent action (Newell, 1980). In other words, an actor should possess a cognitive architecture containing representations, and cognitive and functional mechanisms.

Apart from the notion of physical symbol systems or the classical approach, we explained the importance of two other approaches, connectionism and embodied cognition. Although there are few scientists who claim that these approaches could replace the classical symbolic approach, we argue here that these approaches can complement the classical approach. We acknowledge their importance and show that they can address issues to which the classical approach does not give enough attention.

As elaborated in section 4.5, in order to create a cognitive plausible and a *social* Multi-Agent System, ACT-R needs a task environment that can serve many ACT-R actors. This should be possible without many changes to ACT-R itself. What is needed is a communication interface that allows ACT-R to be plugged into a MAS.

The most important problem that we encountered during the implementation of ACT-R in a social setting was the absence of social mechanisms or representations that allowed for social interaction and coordination. The absence of these mechanisms is not surprising given that ACT-R's interests rest in the domain of the cognitive psychologist.

We are interested in simulations that allow us to study organisations at different levels, the organisational level—interaction between groups or organisations, the social level—interactions between individuals and/or groups, and the individual level—explanation of behaviour of the individual that depends on internal cognitive mechanisms.

In order to explain behaviour at the individual level, we adopted ACT-R. However, in the absence of social mechanisms and social representations in ACT-R, there was a need to draw on other theories. Therefore, we adopted social constructivism and semiotics, as explained in chapter 3, to enrich ACT-R with social constructs/representations, and social mechanisms. This provided us with a stronger explanatory power as regards behaviour at the social and organisational level.

In the next chapter, chapter 5, we explain the design and implementation issues of a new cognitive architecture RBot. The new architecture RBot derives its cognitive plausibility, design and cognitive theories mainly from ACT-R. However, ACT-R is in the first place a cognitive architecture that emphasises cognition rather than social interaction. Therefore, we implement new social mechanisms and social representations allowing for social or normative interaction. In the same chapter, we develop a task environment (MRS) that is able to support

interaction for many RBot actors. RBot actors are plugged into the task environment to allow the RBot actor to perceive signals and signs from other RBot actors. Thus, the next chapter is the description of a design based on a combination of chapter 2—Multi-Agent Systems, chapter 3—the social actor, and chapter 4—the cognitive actor.