Implantation d'un modèle d'attention en **COGENT**

Par

Yi ZHANG

mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maîtrise ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

SHERBROOKE, QUÉBEC, CANADA, Novembre 2004

Le 10 nov 2004,
Date

le jury a accepté le mémoire de Mme Yi Zhang dans sa version finale.

Membres du jury

Mme Hélène Pigot
Directrice
Département d'informatique

M. André Mayers
Codirecteur
Département d'informatique

M. Jean-Guy Dion
Membre
Département d'informatique

(Signature)

M. Richard Egli
Président-rapporteur
Département d'informatique

To my parents and beloved

Sommaire

L'attention est une habilité cognitive qui joue un rôle primordial dans le contrôle des actions. L'attention réfère à l'allocation des ressources pour réaliser une action. L'interférence survient quand plusieurs événements réclament de l'attention. L'objet de ce mémoire est de modéliser l'attention, ce qui permettra de modéliser comment l'attention contrôle les actions humaines. En psychologie, Norman et Shallice ont construit un modèle d'organisation et de contrôle de l'attention. Ce modèle est basé sur deux composants responsables du contrôle de l'action, le "Contention Scheduling" et le "Supervisory Attentional System". Nous présentons dans ce mémoire le modèle au complet mais l'emphase est portée sur le lien entre les deux composants. Des activités de la vie quotidienne sont simulées pour démontrer comment le modèle interagit en cas d'interruption d'une tâche routinière par une nouvelle tâche. Le temps où l'interruption survient est choisi aléatoirement. Le modèle d'attention est alors capable d'ajuster son comportement n'importe quand pendant l'action de la tâche routinière.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Hélène Pigot, my supervisor, for her many suggestions and her constant support during this research. I am also thankful to my co-director, André Mayers, for his guidance and encouragement through my years of study at the Université de Sherbrooke.

Here, I also want to thank the brothers and sisters in Huntingville Community Church. Without their love and help, this work would never have come into existence.

Of course, I am grateful to my parents for their long-term love and support.

Another person I must mention is Mr. Gilles Godin, who is in charge of Student's Financial Aid. This kind gentleman helped me during my difficult struggle searching for finances. I am truly grateful to him for his conscientious work.

Last, but not least, I wish to give thanks to my friends Jinping Ma, Nicolas Chaumont, Dan Baker, Baofeng Yang, Zhonghua Ren and her husband, Jing Liu, Jiang Li, Guang Yang and his wife, for their friendship, care, help and the happiness we shared together during these two years.

Introduction

Historically, technologies were created mainly to help people with physical limitations. Therefore, human lives have benefited from such technological productions as reading glasses, hearing aids, wheelchairs and so forth.

But cognitively impaired diseases, e.g. Alzheimer's disease, are becoming one of the major health problems facing the western world. Patients with Alzheimer's disease are seldom able to live safely and independently. Usually they require at least part-time caregivers, whether they live in medical care centers or in their own homes. Because of growing population of the cognitively impaired population, their families and society are burdened more and more with the demands of care giving. Therefore, techniques emerge for helping this confluence of people, as the times require. To develop such kinds of techniques, fundamentally, one needs to know how human cognitive processes work. The evolution of related fields of study like artificial intelligence and computer science, offer approaches to dabble in the complex area of cognition. Health Smart Home has developed a smart residence for the main purpose of allowing disabled people to stay home with security through connection with a remote health care center [1]. Cognitive assistances are technological ways to make up for the patients' impaired memories or some disabilities. By doing so, cognitive assistants aim to the autonomy of cognitively impaired patients as well as a safe life. Cognitive models are computer models that represent cognitive processes, and can be used to predict some human behaviors through many simulations.

Among human cognitive processes, attention plays a very important role. In psychological literature, attention refers to the allocation of processing resources. It is the ability to focus on a behavior and is responsible for action control. In daily life, people carry out lots of actions to fulfill expected goals. Some actions require attention: novel actions, dangerous ones or actions that need deliberate plans. But routine actions, also called automatic actions, which are well-learned in some environments, require much less attention or can sometimes be executed without awareness. Action, so called behavior, is the act of purpose to accomplish a goal in stages, usually over a period of time. An action is often suspended and resumed later because of interruptions. An interruption is an unanticipated change in the environment while a main action is being performed. Depending on how urgent the interruption is, one may react in three different ways (that will be discussed later) to deal with the interruption. But however the interruption is dealt with, it is under the control of attention. Without attention, it's impossible to switch from one action to another.

The theme of this master's thesis is a model of attention, which gives a view of how attention controls human actions. The model is based on Norman and Shallice's model, a psychological theory, which provides a framework of two-levels of control over actions [2]. The model of attention is implemented in COGENT [3], which provides a visual design environment for cognitive modelling. This thesis is organized as follows: it first introduces an overview of the background; then it reviews the related literature, which includes three parts: 1) the Norman and Shallice's model, on which the model of attention is based; 2) the COGENT programming environment, under which the model of attention is implemented and 3) the works on implementation of Norman and Shallice model; it thirdly presents the model of attention with its objectives, implementation, experimental results and discussions. Finally the paper ends with a brief conclusion.

Chapter 1

Background

1.1 Cognitively Impaired Population

The rising number of cognitively impaired people is one of the greatest health problems in the occidental world. The growth of the Alzheimer's disease is an example: in 1950 in the United States, there were 200,000 cases at most. Within less than 25 years, this number doubled. By 2050 the number of Alzheimer's patients in the world is expected to reach 80 millions [4]. Consequently, the cost of caring for such kind of patients is steadily increasing. The economic spending for care of a patient with cognitive disease is striking. For example, for a single person with Alzheimer's disease, whether at home or in a nursing home, it costs more than $47,000 a year [1].

Moreover, patients with Alzheimer's disease or similar cognitive disorders are not able to live autonomously and safely. They need at least part-time help and care from proxies. However, elder care responsibility increases the physical and financial burdens as well as emotional strains among the caregivers.

To lessen the burdens on the caregivers and to increase quality of care and quality of life for the patients, it is necessary to assist people with cognitive disease in a technological way.

1.2 Health Smart Home

According to the patients' preference, most of them would choose to live an independent way of life in a residential setting with minimum intervention from the caregiver. The second reason for keeping the patients at home is that it is cost diminishing [1]. Thus, the recent establishment of Health Smart Home (HSH), which aims at allowing people with special needs to remain independently in their own homes as well as providing health care services [5]. It is a combination of three research domains: medicine, home networks and information systems. HSH provides an efficient way to develop patient-centred telemedicine and telecare services. Typical clienteles include: disabled and elderly people living alone, people with loss of autonomy, people who need a precise daily medical follow-up and patients with chronic health disease.

1.2.1 Definition

A HSH is such a home that has [5]:

- A number of automatic devices and smart sensors used to ensure patients' safety;
- A local intelligence unit responsible to analyze the data collected by sensors and detect the critical or dangerous situations;
- A connection with a remote control center, which responds to emergency cases and offers an interface between patients and hospital staff or health care centers.

The HSH projects have two objectives: a patient-centered objective and a public health-centered objective. The former aims to improve and enhance the quality, and especially the security of patients' lives by utilizing new technologies. These technologies help them to be more independent in their own environment. The latter aims at the efficiency of public health care services. The patients at home could receive information, advice and supervision from health care services of the hospital through phone calls or net communications.

1.2.2 An Example of HSH

SmartBo is an example of HSH designed for autonomous elderly people who feel insecure living alone at home [6]. The SmartBo project focuses on information and communication technology and computer based solutions for mobility-impaired people, visually and/or hearing impaired people and people with light cognitive disabilities. From its startup, the SmartBo project has continued to develop an ideal smart home for disabled people to live an independent and rich life regardless of their disability.

The project is implemented in a two-room ground-floor apartment including kitchen with dining area and a large bathroom. With no steps or stairs, the entry is easy for the disabled. The entrance door can be opened by remote control.

The installations in the room are divided into two categories: one common to all users, and one adapted for each group or even for each disabled individual. A cable connects all sensors and all devices in the apartment. Sensors are input devices to the cable such as switches, motion detectors, magnetic switches, pressure sensitive switches, current sensing devices, water flow sensing devices etc. Actors are output devices, i.e. relays for switching power, door automation, or controlling lamp dimmers.

Recently, new client/server-based SmartBo control programs are customized upon the clients' need [7]. A great advantage of this client/server-solution is that the client can be chosen for or adapted for maximum benefit of the user. For example, depending on the user's need, a palm computer, or a normal computer is available to the user for his convenience. All computers in SmartBo are linked together in a wireless network.

An alarm is set off in case of water running longer than desired, doorbell or phone ringing, overheated appliances and so on. The alert could be displayed on the computer monitor and even with a voice announcement sometimes.

In addition, some complex functions of the system contribute to the safety of older people. For example, when a person leaves his bed at night, dimmed lamps light the way from bedroom to bathroom. If he doesn't return to bed after a pre-set time, helpers could then be alerted.

1.2.3 Constraints of HSH

While HSH aims for better life for people as well as diminishing family and social burdens, some factors restrict the use of HSH [5]:

- Ethical: their use must take place under the strict control of medicine and ethics;
- Economical: they must be cost effective from the viewpoint of both the patients and society;
- Technological: the equipment used in the homes must be perfectly suitable to the patients' needs, and ensure their safety and privacy;
- Psycho-social: the new technology must be accepted and easily used by all the HSH actors.

1.3 Cognitive Assistance

As mentioned above, the HSH is mainly directed to physical needs. In this section, another new concept is introduced, called cognitive assistance, which implies some contrived ways to help people with cognitive limitations. For people with physical limitations, autonomy depends on the equilibrium between the individual and his environment. In the same way, one's cognitive ability could be improved through environmental context, useful information in the environment and useful ability-supporting technologies. Cognitive assistance, which is defined below, is such an ability-supporting system.

Unlike HSH, cognitive assistance aims mainly to help cognitively impaired people such as Alzheimer's patients, brain injury patients, schizophrenic patients and memory impaired patients.

1.3.1 Definition

Cognitive assistance could be provided both indoors and outdoors. This assistance makes up the patients' deficit to ensure their autonomy as well as their safety. Thus, the action of the system can range from "doing nothing" to drastic operations such as shutting down electricity [8, 9]. In their intelligent habitats, Pigot et al. explained a three-layered infrastructure that cognitive assistance usually comprises.

The infrastructure is divided into three layers:

- Hardware layer: the hardware layer is concerned with many kinds of sensors, devices and effectors, which record the patient's positions, activities and environmental states, offer some advice or provide help directly in some cases;
- Middleware layer: the middleware layer links all the agents of the system in a wireless network, through which these agents can communicate with each other;
- Application layer: the application layer contains two integrated services, one for tele-monitoring, and the other for decision of proper cognitive assistance.

Most people can be in need of cognitive assistance. But here, the objective of cognitive assistance is to develop new systems, mostly computer systems, to enhance the quality of life of cognitively impaired people such as patients with Alzheimer's disease or similar cognitive disorders.

1.3.2 Examples

The Activity Compass is a cognitive assistant that guides disoriented people toward their destination both indoor and outdoor [10]. For example, the compass can guide the patient toward his kitchen when mealtime is overdue. It is developed on the basis of a layered client/server architecture [10], shown as Figure 1.1. In the architecture, the client is a hand held device that handles interaction with user, and the server stores the sensor's readings, constructed models and background information.

Tableau conforme à l'original.

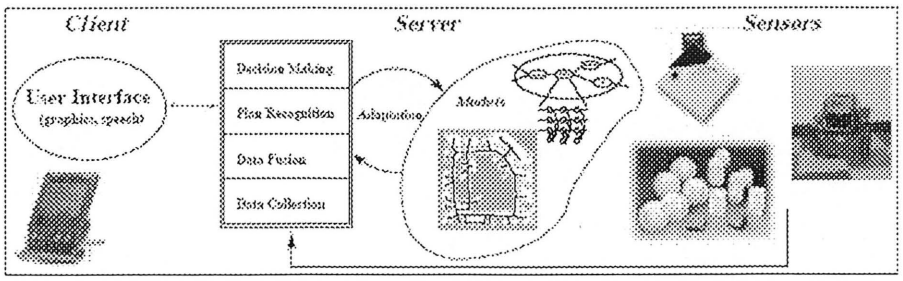


Figure 1.1: The architecture of the activity compass.

The architecture includes several layers to link sensor data to behavior, behavior to plans, and plans to potential interventions:

- Sensor Layer: the layer has a variety of sensors which are installed for different users;
- Data Fusion Layer: the layer extracts information from a continuous stream of data collected by the sensors. Bayes filters are applied in this layer;
- Behavior Recognition Layer: in this layer the user's behavior is estimated based on the output of the Data Fusion Layer;
- Plan and Intention Recognition Layer: this layer performs the task of identifying the user's plans or goals by given a partial view of the user's behavior;
- Intervention Layer: in this layer, the system decides under which circumstances it should initiate an interaction with the user.

The activity compass has to be initialized before it can work effectively. Firstly, the compass needs to construct a high-level model of the patient's activities by recording the person's habitual activities coupled with background information and sensor readings. Secondly, plan recognition is essential for the compass to make sense of the patient's current actions. Finally, the compass needs to detect alternative pattern under uncertainty cases requiring appropriate reasoning.

1.4 Cognitive Modelling

1.4.1 Outset of the Issue

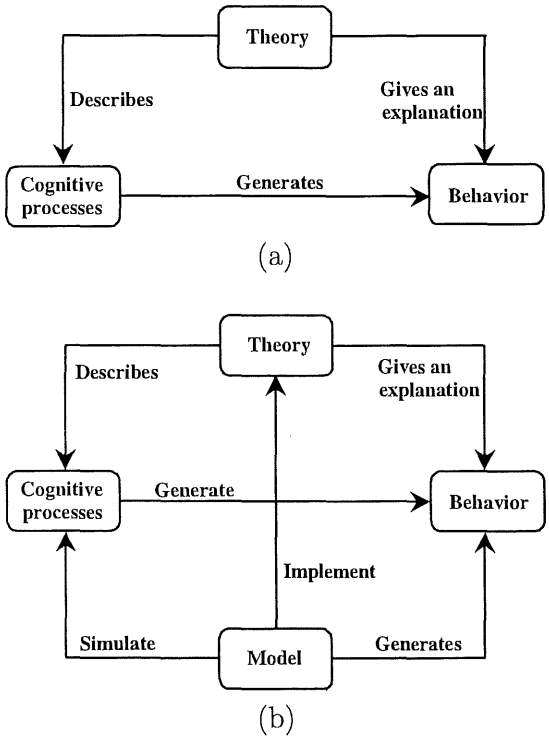


Figure 1.2: (a) The classical relations between a behavior, underlying cognitive processes and a theory of those processes; (b) The relation between a cognitive model and a behavior, a theory, and cognitive processes.

As its name implies, cognitive modelling is the development of computer models, which represent cognitive processes, and use of such models to simulate or predict human behavior. A model is a representation of a real object, for example, a model in architecture may be a small paper-house to show the real house's appearance or to evaluate its structure. Similarly, a computer model is also a representation but using specified computer language. This study concerns two issues: 1) the contents of the model and 2) the way the model is designed. The first issue is discussed here, the second one will be discussed section 1.5.

1.4.2 The Role of Cognitive Modelling

Cognitive modelling is based on a concern of three entities: behaviors, cognitive processes underlying them and theories of these cognitive processes. Behaviors are the actions done by the subject with some purposes. Cognitive processes of the subject, such as long term memory, generates behaviors. A well suited theory describes the cognitive processes and consequently gives an explanation for the behavior generated by those cognitive processes. Figure 1.2(a) shows their relations. Computer simulation techniques allow cognitive models to be built, thus, an extra entity is added to the Figure 2.1, demonstrated as Figure 1.2(b). A model plays the role of generating behaviors, implementing a theory and simulating a cognitive process.

1.5 Cognitive Architecture

1.5.1 An overview

A cognitive architecture refers to a particular set of conceptual structures, tools, techniques and methods that support the design and the construction of cognitive models. A cognitive architecture embodies the more general structures and mechanisms out of which could be made a model of individual cognition in a certain situation [11]. A cognitive architecture is an integrated system capable of intelligence supporting, which may comprise a short-term memory, a long-term memory, perceptual and motor subsystems, some learning mechanisms, and so on. Shown as Figure 1.3, a cognitive model is an instance of a cognitive architecture in a specified domain.

Cognitive architecture, such as Soar [12,13], ACT-R [14,15], CAP [16] and EPIC [17], models human behaviors via different approaches. Soar and EPIC are symbolic architectures, CAP is a connectionist architecture and ACT-R is a hybrid architecture. Among these architectures, ACT-R is currently the most influential one because it offers whole mechanisms in generalization which is domain independent, and because it approaches

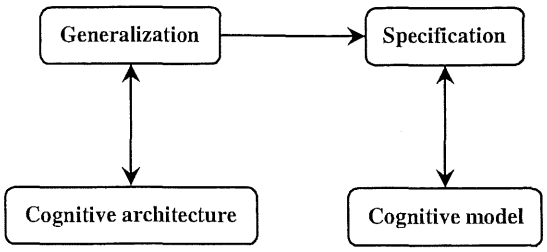


Figure 1.3: A cognitive model is an instance of a cognitive architecture in a specified domain.

both from symbolic and from connectionist.

1.5.2 ACT-R

1.5.2.1 Introduction

ACT-R is a production-rule based cognitive architecture, which is implemented in Common LISP language. It intends to simulate the human behavior in a generalized way. It simulates the acquisition of knowledge and human actions influenced by the environment.

Since ACT-R is a cognitive architecture, it covers a wide range of human cognitive tasks, focusing on problem solving, learning and memory. It has been previously applied to modeling such tasks as solving the standard Tower of Hanoi problem, memory for text or for lists of words, language comprehension, communication, task switching and aircraft controlling.

In ACT-R, knowledge is represented by two types of memory: declarative memory and procedural memory. Chunks are the atomic components of declarative memory. The basic unit of procedural memory is production rule. A chunk is defined by its type (e.g., birds) and its slots (e.g., color or size). A production rule is a condition-action pair. The production rule specifies an action to be executed when a condition is satisfied. The condition consists of a specified goal and a set of chunks while the action usually creates or modifies some chunks. Thus procedural memories manipulate chunks to produce

cognition. Two buffers are defined in ACT-R: the goal buffer (to specify the actual state of a goal) and the retrieval buffer (to store information retrieved from declarative memory). Four other buffers are defined in perceptual motor, the addition of ACT-R. These buffers are the visual buffer, the manual buffer, the aural buffer and the vocal buffer. They all correlate to the human perception and will be discussed later.

1.5.2.2 Mechanisms

In this section, main mechanisms in ACT-R are discussed. Through these mechanisms, ACT-R manipulates chunks and production rules, sub-goals, perceptual motor, and learning ability.

Activation and Latency

Each chunk of the declarative memory is associated with an activation value. This value is evaluated at each cycle of processing. It depends on the basic activation, the strength of association with other chunks, the similarity between each part of the chunk and each part of the retrieval condition. Therefore, the activation value of the chunk i is expressed by the formula below:

$$A_i = B_i + \sum_j W_j S_{ji} + \sum_k QM_{ki} + \varepsilon_1 + \varepsilon_2. \tag{1.1}$$

It is computed as following:

The base-level activation B_i reflects the recency and frequency of practice of the chunk. It is influenced by two parameters:

1. n_i, the number of times that the chunk i is practiced;
2. L, the real time elapsed in seconds.

$$B_i = \log(n_i/L - d) - d\log(L), \tag{1.2}$$

where d is a parameter set by the user at the beginning of the simulation. Thus B_i increases with practice and decreases with time.

The sources of Activation: computed as the summation below (Sum. (1.3)):

$$\sum_j W_j S_{ji}. \tag{1.3}$$

The elements j being summed over are the chunks which are in the slots of the chunk i. W_j is the amount of activation from source j. S_{ji} is the strength of association from source j to chunk i. It is set using this equation (Eq. (1.4)) when chunk j is in a slot of chunk i:

$$S_{ji} = S - \ln(fan_j), \tag{1.4}$$

where S is a parameter to be set with the maximum associative strength; fan_j is the number of chunks in which j is the value of a slot plus one for chunk j being associated with itself.

The final term that influences the retrieval of the chunk i is the partial matching component. Expressed as the following summation (Sum. (1.5)):

$$\sum_k Q M_{ki}. \tag{1.5}$$

Specification elements k: the matching summation is computed over the slot values of the retrieval specification; match scale Q: this reflects the amount of weighting given to the similarity in slot k. By default this is a constant across all slots with the value of 1; match similarities, M_{ki}: the similarity between the value k in the retrieval specification and the value in the corresponding slot of chunk i.

To simulate the chunk retrieval in a more realistic way, it should not be completely deterministic. Therefore, two types of noise have been introduced. The permanent noise ε_1, which is given by equation is set at the creation of each chunk, while ε_2, the transient noise, is set each time the chunk is retrieved. Both noise values are generated according to a logistic distribution characterized by a parameter s.

Probability of Recall

A chunk is recalled only if its activation value is over a threshold, which is named utility threshold. Among the chunks whose activation value is over the utility threshold,

the probability of choosing a chunk i is given by the equation:

$$\mathbf{prob}(i) = \frac{e^{A_i/\sqrt{2}s}}{\sum_c e^{A_c/\sqrt{2}s}}, \tag{1.6}$$

where A_i is the activation value of chunk i, A_c is the activation value of any competing chunk c. s is the parameter s of the noise.

ACT-R also simulates the retrieval time for each chunk. The rule is quite simple: the higher the activation, the faster the chunk will be retrieved. The retrieval time T_i is given by the following equation (1.7). Where F is another parameter to be estimated, called the latency scale.

$$T_i = Fe^{-A_i}. \tag{1.7}$$

Utility Learning

The utility learning is used to determine which production rule will be chosen to fire if several of them satisfy the same conditions. Each production is associated with an utility. It reflects how much the production is expected to contribute to achieve the system's goal. The one that has the highest utility U_i will be chosen.

This utility is an estimation based on experience. It increases with success and decreases when the cost increases.

The success is evaluated through the estimation of the probability of success P_i. It is computed as following:

$$P_i = \mathbf{successes}/\mathbf{totaluse},$$

where **totaluse** is the total number of times the production rule has been tried during the experiment and success is the number of successes of this production along its total use.

The value of the cost C_i represents the average amount of time, counted in seconds, spent to execute the production rule i, then:

$$C_i = \mathbf{totaltime}/\mathbf{totaluse},$$

where the **totaltime** is the cumulative time over each time the production rule is used (**totaluse**).

As the cost is counted in seconds, the value of the objective G is also counted in seconds. It has a default value of 20 seconds, which can be changed by the user. This value of the goal times the probability of success represent an estimation of success of the production rule i. So, the utility U_i for the production rule i is:

$$U_i = P_i G - C_i + \varepsilon. \tag{1.8}$$

Because it is more realistic to maintain a level of uncertainty, a noise ε is always added to simulate randomness. It follows the same logistic function as either ε_1 or ε_2 in the computation of the level of activation A_i.

Like the choice of a chunk, the choice of a production rule is performed in a probabilistic way, called the probability of recall of the production rule. It is given by the equation:

$$\textbf{Probability}(i) = \frac{e^{U_i/\sqrt{2}s}}{\sum_j e^{U_j/\sqrt{2}s}}, \tag{1.9}$$

where the summation is over every competing production rule, including utility threshold as if it were a production rule. s is the parameter s of the noise.

Production Rule Learning

A new production rule can be built from two old production rules by applying succession into a single rule. This process of acquiring new production rules is called production composition. The basic idea is to combine the tests in two conditions and combine the two actions into one that has both effects. The composition is based on buffers, because usually the conditions involve buffer tests and the actions involve buffer transformations. The complications of production composition can be divided into three cases according to whether the old production rules request a new goal. Many technical details are needed to make sure that the proper references are built to variables and

constants in the composed productions. If the reader is interested, please check them at: `act.psy.cmu.edu/ACT-R$_-$5.0/unit9.doc`

Conflict Resolution

After a new production (called **New**) is composed from two old productions (**Old1** and **Old2**), a fact is that whenever **New** could apply **Old1** could also apply, otherwise, it does not mean that whenever **Old1** could apply New could also apply, because **New** might be specialized. The choice among the **New**, the **Old1**, and whatever other productions might apply will be determined by their utilities. As discussed before, if **New** could apply, **Old1** could apply also, that means **New** and **Old1** have the same probability. Moreover, since **New** is created by merging **Old1** and **Old2**, it is more specified than **Old1**. Therefore, **New** will have less chance to be successful than **Old1** will. A parameter α has been introduced (called initial experience) to reflect the setting of the successes or failures for new production, whose value is ranged from 0 to 1. This expresses the fact that **New** is more specific than **Old1**.

The new production rule is supposed to be better than the old one. The problem is that because of the lower initial utility than the older production rule (scaled down by α), there is less probability for ACT-R to choose them. This problem is overcome by the noise parameter, which introduces some irregularities into the computation of utility functions. Sometimes the new merged production rules will be chosen. Because they usually execute a complex task faster, their utility will increase and they will tend to be used more and more.

Subgoals

ACT-R offers a mechanism of subgoals. With it, ACT-R enables the subgoal hierarchical processing. The main idea of subgoals is the division of a complex goal into several different subgoals, each of which is considered as a child of the original goal (called parent).

Each subgoal is set by using the command "`+goal> expression`". This expression

assigns a certain task to the subgoal. After the subgoal finishes its task, it may go back up to its parent. To retrieve the original goal, use the command "`+ goal>`" and set "`parent = goal`". Let us pay more attention to the latter command. By setting a goal to the "`parent`", it enables ACT-R to create a hierarchy of subgoals. That means whenever considering of a subgoal, its situation must be taken into account, because this subgoal may also be a parent of other sub-subgoals.

Following is a simple tree to show the generations of goal and subgoals:

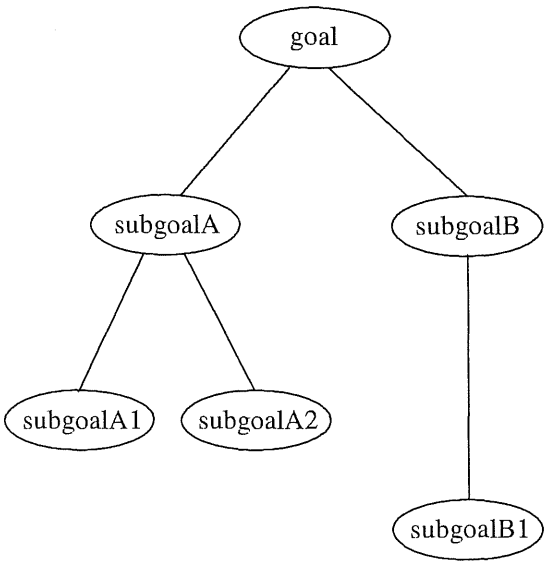


Figure 1.4: A family tree of a goal and its subgoals.

In the tree structure, **subgoalA** and **subgoalB** are two subgoals of **goal**, and at the same time, they are also parents. **SubgoalA** has two sub-subgoals: **subgoalA1** and **subgoalA2**; and **subgoalB** also has **subgoalB1** as its subgoals.

Perceptual Motor

Overview

Around 1997, Perceptual Motor was added. Together with ACT-R, it is ACT-R/PM. Perceptual Motor, in fact, plays an important role since its appearance, because

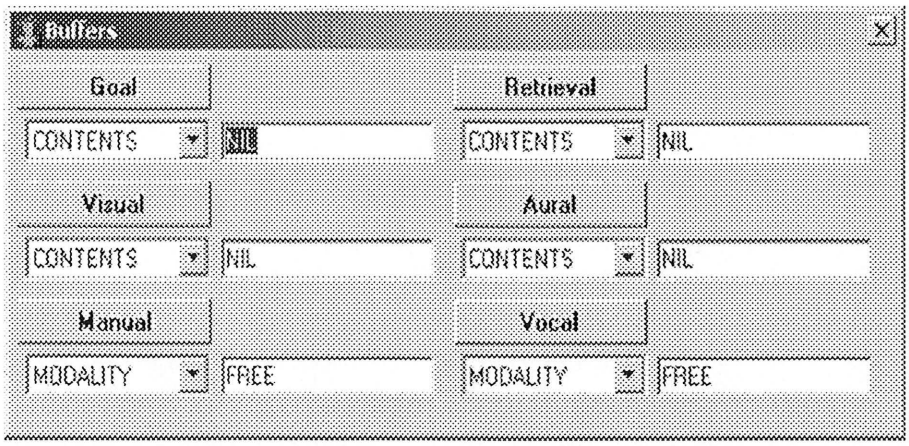


Figure 1.5: Perceptual motor includes four buffers: Visual, Manual, Aural and Vocal buffers.

it mediates the communication between cognition and the external environment. Shown as Figure 1.6, is an overview of the ACT-R/PM system [18]. As we discussed before, production rules manipulate chunks through buffers. Four of these buffers belong to the perceptual motor. They are respectively: the visual buffer, the manual buffer, the aural buffer and the vocal buffer (see Figure 1.5). Stimuli come from these four different sources, and correspondingly there are different modules in Perceptual-Motor layer to handle some certain aspects of perception and action.

Attention

In ACT-R, attention and focus are two different actions. It means in order to focus on another location, the model has to take two steps: first move attention and then focus on the new target. For example, it is easy for a person to notice a motion, like a mouse running on the floor, while focusing elsewhere at the same time, on the TV for example. Despite the fact that the attention is on the TV, the vision area is wider than the screen only. That's why it is possible to notice what happens around. This illustrates the fact that seeing something is different from focusing on it. In ACT-R, this aspect is taken into account: in order to focus on a location, the model has to look at this location first.

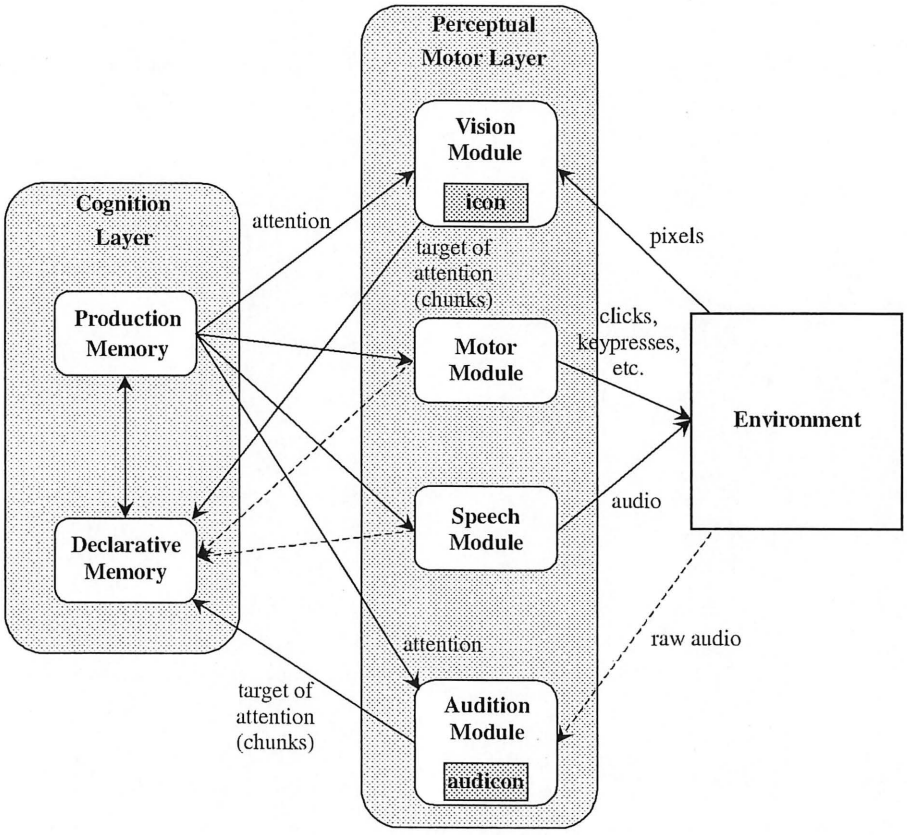


Figure 1.6: An overview of ACT-R/PM system.

Chapter 2

Review of literature

2.1 Introduction

This chapter is divided into four parts. The first section gives the definitions of some important concepts in the current research. Then the second section introduces the Norman and Shallice model, on which the model of attention in the current research is based. The model consists of two mechanisms, the Contention Scheduling and the Supervisory Attentional System, which together explain how attention can control human actions. Next, the third section introduces the COGENT, an implemental environment for cognitive objects, under which the model is built currently. Finally, the fourth section reviews some previous works that have been done by others on implementation of the Norman and Shallice model.

2.2 Concepts

2.2.1 Action

Action, so called behavior, is defined as the act of purpose to accomplish a goal in some stages, usually over a period of time. Two kinds of action are considered: automatic

behaviors and willed behaviors. The former, also called well-learned actions, refers to the activities that can be carried out with diminished awareness; the later is the performances based on will or attention. Whether the purpose is conscious or not, the action is directed by a goal that is satisfied at the end of the completed action. Some goals are simple and do not need to be broken into sub-goals. In contrast, complex goals are divided to create simpler ones, each of them representing an action to be performed. Accordingly, the action to achieve a complex goal should be decomposed into several sub-actions, each of which fulfills a sub-goal. This hierarchical structure could be presented by a tree where the action and the goal levels alternate [19].

2.2.2 Attention

Attention refers to the allocation of processing resources. It is the ability to focus on a behavior. Some actions require attention: novel actions, dangerous ones or actions that need deliberate plans. But routine actions, called automatic actions, which have been performed several times in the same conditions, require much less attention. In some cases when unanticipated interruptions or unexpected environment changes occur, the attention rises to cope with the new situation. Therefore, the same action may belong to the automatic or to the willed action depending on the environment in which it is performed. For example, driving a car is considered as an automatic action. Every day, one drives from home to office and back. While driving his well-known route, the driver can talk or listen to the radio without much concentration on his action. But if a hazardous situation takes place on the route, like a traffic jam or a child crossing the road, the driver stops his conversation and turns his attention to the road. In addition, if the journey is new, the driver needs to pay attention to the road situation, the traffic lights, the road signs and so forth. In these cases, the action of driving is considered a willed action.

2.3 Norman and Shallice Model

2.3.1 An Overview

Norman and Shallice provided a framework that comprises two mechanisms to control the two kinds of actions defined above, the Contention Scheduling (CS) and the Supervisory Attentional System (SAS) [2]. CS is responsible for well-learned actions, and SAS is required when the action is new, or when the action itself is dangerous, or when the plan must be modified according to an unanticipated environmental change. Actions are represented as some schemas in memory. CS controls actions directly by selecting from among competing schemas. While SAS controls actions via biasing the selection process. Figure 2.1 is an overview of the architecture of these two levels of control.

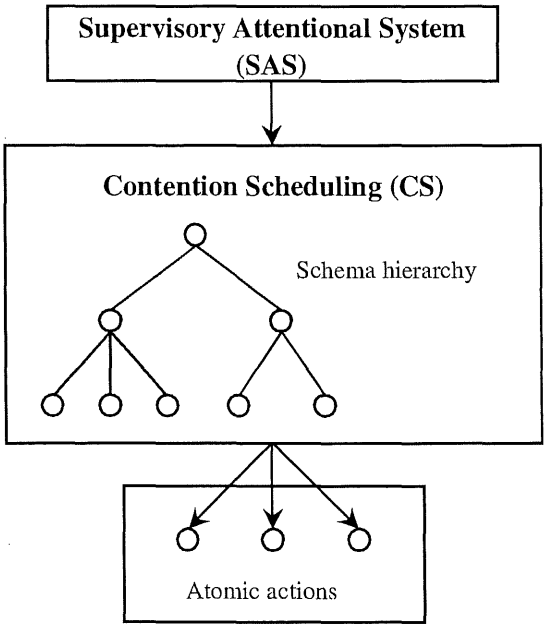


Figure 2.1: The architecture of the two controlling levels: CS and SAS.

Most of the time, CS makes the selection alone. SAS oversees the processing of CS

and interferes when necessary. Since a complex action is usually divided into a set of sub-actions, the schemas for these sub-actions are organized hierarchically in CS according to different levels of actions. Selections among these schemas are mainly based on their activation value. Each schema is assigned to an activation value. This value varies over time during procedure of an action's performance. When a schema's activation value is higher than a threshold, it is selected. A selected schema will lead to either an internal action or an external action in the world.

In addition, the model contacts with its environment. Some events in the environment can trigger appropriate schemas. Consider a simple action as "picking up a pen". This action can be represented by a set of component schemas like reaching the pen, holding it and picking it up. Triggered by the perception of the pen on a table, the source schema of "picking up a pen" is set up by an activation, which in turn activates the component schemas in a sequence and results in choosing the proper arm, hand, and finger movements to get the pen. Such a sequence is called a horizontal thread. A simple horizontal thread is shown as Figure 2.2. Selection of component schemas is determined mainly by how well its trigger condition matches the contents of the trigger database.

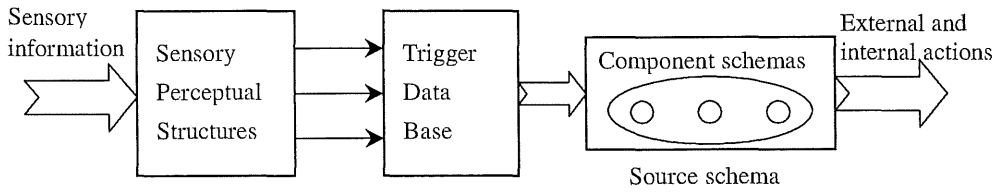


Figure 2.2: A horizontal thread for an automatic, well-learned behavior.

In many situations, several actions can be performed synchronously, each of them is specified by its own horizontal thread. Different component schemas may all access a common memory database, or need use the same sources (i.e., the same hand). Some means may become necessary to provide resolutions for conflict under these situations. As a result, the different threads may interact with one another. Shown in Figure 2.3, the lines between different schemas represent their interactions. It is similar to the concept of multi-thread process in computer science.

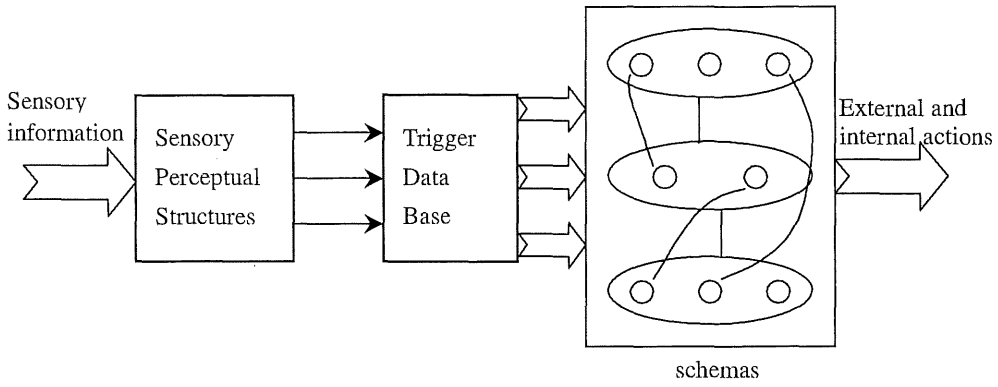


Figure 2.3: Three horizontal threads may take place at the same time.

In some cases, attention is required for a particular action either because some critical or dangerous situations are involved or because the action itself is not well-learned or because some unexpected issues interrupt the action. In these cases, the relevant horizontal threads are not sufficient for controlling the action. Therefore, the vertical threads, shown as Figure 2.4, take over. In fact, these vertical threads take control over an action only by providing additional activations, as adding activation for the desired schemas and decreasing that of the undesired ones.

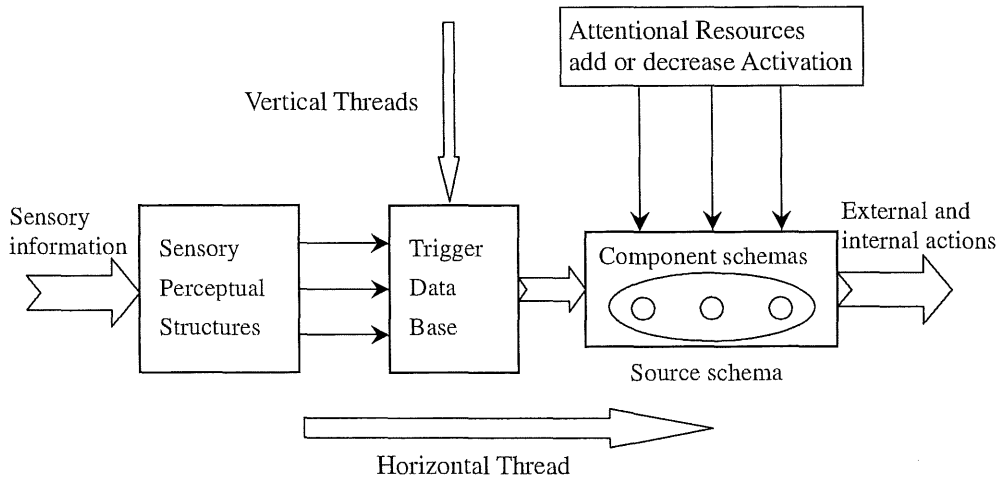


Figure 2.4: Vertical threads provide additional activations when attention to particular action is required.

2.3.2 Contention Scheduling

The CS mainly comprises two parts: the schema hierarchy, within which schemas are competing and the schema selection mechanism, which is dedicated to the selection between these schemas. The following describes them in details.

2.3.2.1 Schema Hierarchy

The core of CS is the schema hierarchy and within which a set of schemas are competing. Shown in Figure 2.1, actions are represented by a number of memory schemas. When an action is complex, it may be broken down into a sequence of sub-actions. Schemas are organized into a hierarchy according to different levels of actions, within which the source schema represents the action and the component schemas represent its sub-actions. A schema controls an action, either an internal processing or an external movement of effectors. The source schema is serving as the highest order control and its component schemas can be activated via the source schema.

There are three states of a schema: dormant, activated and selected. A schema is dormant when it rests in the memory, playing no role in current processing. For example, the schemas of cooking are usually dormant during the individual's sleep. A schema is activated when it is set up with an activation value. Whenever this value is sufficiently high, it is selected. A schema's activation is influenced by several factors:

- A top-down influence, also called internal influence, which is from the source schema or directly set by the SAS;
- An external influence, also called environmental influence, which is an input from the environment. For example, a reachable spoon can excite a schema that represents an action like "pick up a spoon";
- A lateral influence, which comes from interactions with other schemas;
- A self influence, which is opposite to the lateral influence.

2.3.2.2 Schema Selection Mechanism

The schema selection mechanism is in charge of selection among relevant schemas in order to carry out an action properly. The selection of a schema is according to its activation value and a pre-assigned threshold. Whenever a schema's activation value exceeds the threshold and is higher than that of any other competitor schemas, it is selected. Under some situations, several schemas may have an activation value which is higher than the threshold at the same time. In this case, these schemas compete with each other and the one which suits the situation most can get more activation and finally be selected. This procedure, therefore, is similar to the retrieval of chunks in ACT-R [14]. As mentioned in Chapter 1, when ACT-R processes knowledge, it has to determinate which chunk is to be retrieved. Once the activation of each chunk is computed, every chunk can be potentially chosen with a probability which depends on their activation. First, if the activation is below a value, which called the utility threshold, the chunk has no chance to be chosen at all. In case it is over this threshold, it provides a chance of that chunk to be retrieved. But only the chunk which most accurately fits the conditions will have the highest activation level and will be retrieved.

The selection depends upon three effects: horizontal threads, vertical threads and current environmental conditions. The horizontal threads determine the organization of a particular action sequence. For example, to make a coffee, one can add sugar before adding coffee powder or vice versa. The vertical threads determine the biases acting on the selection process. Usually sugar could be taken from a sugar bowl or from packets in the same example. But if the sugar in the bowl is dirty, the vertical threads will inhibit it and therefore give more chance for the sugar in packets to be selected. The trigger conditions determine the appropriate timing for schemas. For example, an opening milk carton may trigger the action of adding milk to the coffee or the action of closing the carton.

2.3.3 Supervisory Attentional System

The SAS controls actions under non-routine situations. Norman and Shallice suggested that SAS oversees the performance of action and participates in control by creating schema or by modifying activation values when needed [2]. When an action is novel, or some dangerous situations are involved, or some unexpected interruptions shows up, attention is required. Attention is put in from SAS to CS through the vertical threads, which controls over schemas by modifying the schemas' activation values, either increasing the values of desired schemas or inhibiting the values of improper schemas according to the different situations. Shallice and Burgess addressed three stages of SAS to respond to uncommon situations [20]:

- The construction of a temporary new schema;
- The implementation of this temporary new schema;
- And the monitoring of the schema execution.

Coping with a novel situation involves a variety of different types of processes operating over these three stages. The key element is the construction and implementation of a temporary new schema, which takes the place of the source schema, and which in turn can control the lower-level schemas to provide a plausible procedure for action control over the novel situation. The 8 processes involved in these three stages are displayed below. Numbers in parentheses are specified by Shallice and Burgess [20].

Stage 1 (the construction of a temporary new schema)

- The process (4) generates strategies for coping with the current situation automatically by just following a sense of the preceding method of tackling the situation;
- The process (5) generates strategies by using problem solving, which include a series of phases like problem formation or orientation, the phase of attempting to find a solution, the assessment of a solution and the phase of checking;

- The process (6) leads to goal setting, which is required in the phase of problem formation to provide the criteria for later assessment of the solution attempt;
- The process (7) prepares a strategy and plan for a later time;
- The process (8) provides raw material of related experiences for confronting a new situation via episodic memories retrieval.

Stage 2 (the implementation of this temporary new schema)

- The process (1) implements the temporary new schema, which has been constructed in the stage 1.

Since in a new situation, a schema may not be triggered automatically, a special working memory, which is used to hold the temporally active schema, is required to implement the operations of the temporary new schema.

Stage 3 (the monitoring of the schema execution)

- The process (2) monitors how well the processes in stage 1 work in a new situation;
- The process (3) rejects or alternates the temporary new schema.

2.4 COGENT: the Cognitive Modelling Environment

COGENT stands for Cognitive Objects within a Graphical EnviroNmenT. It is a graphical environment for computational modeling of cognitive processes [3].

Typically, cognitive modeling is performed either in a cognitive architecture as ACT-R [14], or directly in AI programming language as Prolog [21]. COGENT is designed to allow psychologists with little or no computational expertise to develop their own computational models. It provides an integrated set of tools to simplify the process of developing and evaluating computational models of high-level cognitive processes.

The basic system provides the modeler with a sketch pad on which the model can be drawn as a box and arrow diagram. This box and arrow notation builds upon the concepts of functional modularity (from cognitive psychology) and object-oriented design (from computer science). Different types of box are used to represent different types of component. Each box represents an object. Different types of arrows are used to indicate different types of communications between boxes. COGENT is not an architecture like ACT-R [14] or SOAR [12,13]. It does not embody any specific assumption about control processes and it does not contain any pre-specified learning mechanisms. COGENT is most appropriate for the development of cognitive models which are not clearly rooted in any specific architecture. However, as an environment, COGENT provides facilities for implementing complete architectures or learning mechanisms within models of varying complexity.

COGENT is not domain specified, it is intend to be as flexible and domain general as possible. By now, COGENT has been used to develop models in the domains of problem solving, memory, reasoning decision-making, categorization, mental rotation, routine action control and executive process control.

COGENT is a powerful computational modeling system, which is designed to simplify rigorous development and testing of cognitive models, and to aid data analysis and reporting. It provides:

- Memory buffers to store information;
- Rule-based processes to operate on the information stored in the memory buffers;
- Connectionist networks to represent connectionist object types;
- Inter-module communication links to relate compound boxes by means of arrows;
- I/O devices to send or collect data and objects during model execution.

COGENT also has a good testing environment which allows the users to test their model step by step or in a whole. During the execution of a model, the traffic between

Tableau conforme à l'original.

components could be viewed through messages of each component's window. Following, some special components of COGENT will be highlighted in details.

2.4.1 The Visual Programming Environment

COGENT offers a visual programming environment out of which models may be developed, edited, executed and tested. Especially, the programming environment allows users to create models in a box-arrow notation. Figure 2.5, from the COGENT tutorial [22], shows a box-arrow diagram of a production system. In the diagram, the rounded rectangle represents a buffer, the square rectangle is compound box, the hexagonal box represents a process, the solid-end arrow and the bare-end arrow are representing the read arrow and write arrow respectively.

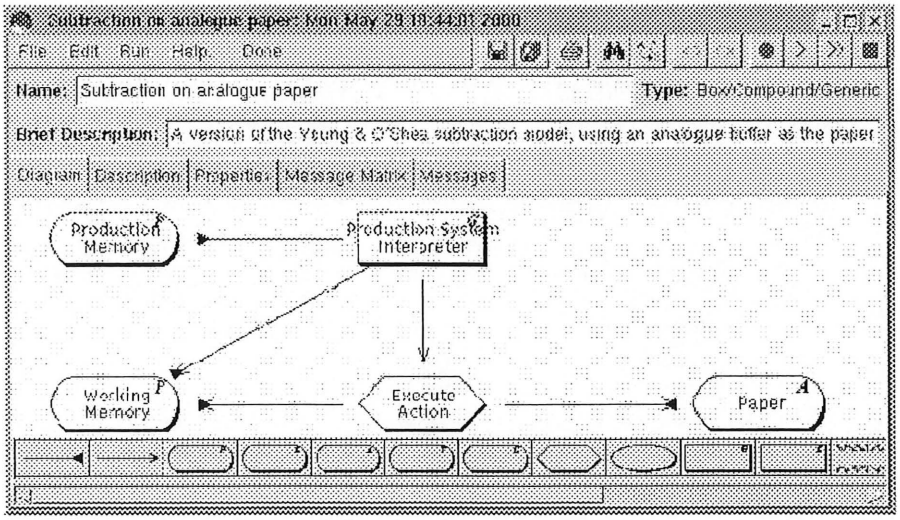


Figure 2.5: A box-arrow diagram of a production system.

The graphical model editor offers a number of standard types of component, such as kinds of buffers, rule-based processes, networks, compound boxes and so forth, which will be described in details in section 2.4.3. The model shown in Figure 2.5 uses two propositional buffers (Production Memory and Working Memory), a rule-based process

(Execute Action), an analogue buffer (paper), and a generic compound box (Production System Interpreter). These components communicate via reading from and writing to one another.

2.4.2 Standard Functional Components

2.4.2.1 Memory Buffers

A memory buffer is a general information storage device, which is appropriate for both short-term memory (i.e. as in modeling working memory) and long-term memory (i.e. as in knowledge bases). Five types of buffer box are available:

- Propositional buffer: it is the most basic type of buffer supported by COGENT. No restriction is set on the types of items stored in a propositional buffer;

- Stack buffer: a stack is a data structure that allows last-in, first-out (LIFO) access. Stack buffers are more often used to maintain a sequence of goals and sub-goals. Only the top-most sub-goal could be operated until it is achieved and be popped off the stack, and then the next sub-goal is revealed. These operations will continue till the stack is empty, that means the final goal is achieved;

- Analogue buffer: this type of buffer is specified in COGENT to contain and manipulate graphical representations, such as points, lines, circles, boxes, and so on;

- Table buffer: in a table buffer, data are represented in a two-dimensional table, thus, the messages are formed like **data (row, column, value)**;

- Graph buffer: in a graph buffer, data are represented in several graphical formats like histogram, line-graphs or scatter-plots.

2.4.2.2 Networks

Three types networks are available in COGENT, the feed forward network, the interactive activation network and the associative network.

- Feed forward network: it provides a general two-layer feed-forward network capability. It is an object which consists of a set of input nodes and a set of output nodes, together with a set of weighted connections between the nodes. This kind of network is able to map input vectors (of the specified width) to output vectors, and to learn input/output correspondences;

- Interactive activation networks: it contains nodes with associated activation values, which should be real numbers. Interactive activation networks normally co-exist with a process which sends appropriate excite messages to control the node activations:

  send excite(**NodeName**, **Excitation**) to MyNet.

  In such a message, **Excitation** should be a numeric quantity which specifies the level of excitation to apply to the node. Another operation available on networks is matching. The activation value of a node may be queried by matching against the node's name:

  node(**Name**, **Activation**) is in MyNet.

  The above code segment may occur in the conditions of any rule or condition definition that can read MyNet. Nodes within an interactive activation network may be created or deleted. A nodes can be excited (with a positive number) or inhibited (with a negative number). COGENT offers a special purpose dynamically updated viewer, so called Activation Graph View, allows activations to be monitored during model execution;

- Associative network: it consists of a set of nodes joined by weighted connections. Nodes may be fully connected, as in interactive activation networks. And weights

between nodes may be learnt as in feed-forward networks in response to training signals. Associative networks may be configured by setting the network's size, learning rule, learning rate, activation function, etc. Associative networks may be sent two types of messages:

send learn(**Vector**) to MyNet, messages are used to train the network; send excite(**Vector**) to MyNet, messages are used to activate nodes within the network.

2.4.2.3 Compound Boxes

Compound box is a box that contains a set of sub boxes. The behaviors of the set of sub boxes and their interactions determine the entire behavior of the compound box. The system defines two kinds of compound box. One is the General compound box and the other is the Subject compound box. As the name implied, the former is general, it may contain any types of boxes and has no restriction on those boxes; the latter, has some constraint on its sub boxes, that is they must make sense themselves. Therefore, a subject compound box can not have a sub box as general compound box or some other data boxes, meanwhile have another subject compound box as its content.

2.4.3 The Rule-Based Modeling Language

In COGENT, processes are specified in terms of a powerful but complex rule-based modeling language. Each rule is a condition-action pair. The condition includes logical operations, which typically involved matching elements in buffers and whose outcome may be true or false. Action passes a list of messages to their destinations when the rule's condition is true. Following is an example of a rule in COGENT. This rule fires when there is a word in the short-term storage. And by firing, it adds the word into the long-term storage:

```
IF Word in STS
THEN add Word in LTS
```

COGENT's representation language is borrowed from Prolog [21], a programming language developed for AI applications. The representational units include:

1. Atoms: which are sequences of letters or other characters with a number of occurrence limitations;
2. Numbers: which are integers or real numbers;
3. Lists: which are sequences of information;
4. Compound terms: which are used to represent information with internal structure, i.e., "Salt is a seasoning" could be represented as seasoning (salt);
5. Variables: which are unknown or non-fixed information.

COGENT also permits users to define their own conditions using Prolog.

2.5 Implementation of Norman and Shallice Model

2.5.1 Contention Scheduling Implementation

Cooper and Shallice have implemented a detailed computational model of CS [19]. It comprises three functional networks: Schema Network, Object Network and Resource Network. The heart of CS is the Schema Network. The other two networks serve to model object representations and resource representations. In addition, a selection process oversees the schema network and interfaces with the object, resource, and motor system.

2.5.1.1 Schema Hierarchy

In the Schema Network, nodes are organized hierarchically according to different level of actions. Since an action is an act of will, each schema is also associated with a goal. Thus an action is represented in a hierarchy within which schema levels alternate

with goal levels. Figure 2.6 shows an example of such hierarchies, which represents an action and its schema/goal organization.

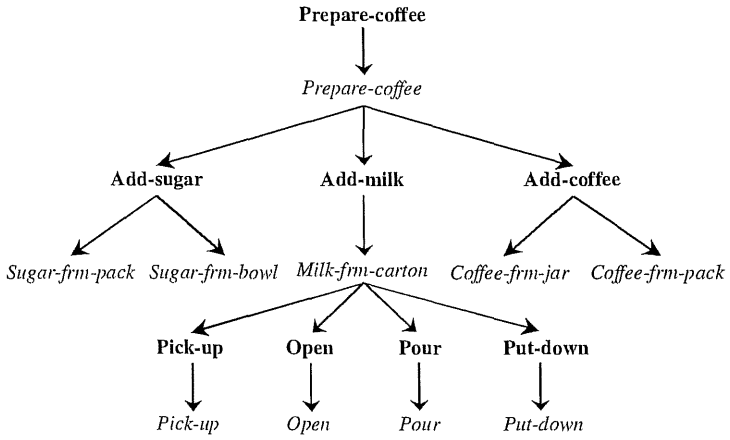


Figure 2.6: Schema/goal organized for the action of coffee preparation. Goals are printed in bold type and schema in italic type.

To fulfill a goal, at least one schema is proposed. This schema, in turn, may be associated with a set of subgoals. For example, in Figure 2.6 the goal **prepare-coffee** is fulfilled by one schema *prepare-coffee*. While to execute the schema *prepare-coffee*, three goals are to be achieved: **add-sugar**, **add-milk** and **add-coffee**. In addition, the goal **add-coffee** could be fulfilled either by the schema *coffee-frm-pack* or by the schema *coffee-frm-jar*. Therefore, within the schema/goal hierarchy the components of goal nodes are disjunctive while the components of schema nodes are conjunctive. Thus if the parent is a goal, one (and only one) of the children has to be executed. If the parent is a schema, all the children have to be fulfilled.

2.5.1.2 Activation Value

Each schema is also associated with an activation value which varies over time. At any time, the net influence on schema activation is given by a weighted sum of fours sources [19]:

- Environmental influence: schema activation may be affected by the presence or absence of appropriate triggering situation in the environment;
- Internal influence : also called top-down influence, which is from SAS or high-level schemas;
- Lateral influence: this influence ensure that competitive schemas inhibit each other;
- Self influence: a self influence generally works to excite schemas;

In addition, nodes in Object Network and Resource Network are also associated with activation values. Cooper and Shallice report the detailed calculations of activations of all networks and how the four influences contribute to the activations in their implementation of CS [19].

2.5.1.3 Selection Mechanism

The Selection Process connects the Schema Network with Motor System and oversees their status. A schema is selected when its activation exceeds a threshold. This selected schema in turn excites its children schemas. Selection of a lowest-level schema will lead to an action in the motor system.

2.5.2 Supervisory Attentional System Implementation

2.5.2.1 Internal Processes of SAS

Shallice and Burgess [20] have divided the internal processes of SAS into three stages: 1) the construction of a temporary new schema; 2) the implementation of this temporary new schema; and 3) the monitoring of the schema execution. Shallice and Burgess specified 8 processes in these three stages [20]. These internal processes of SAS are similar to the usual ones that are always addressed in artificial intelligence. Therefore, Glasspool [23] proposed to use an artificial intelligence agent, Domino, as an approach for implementing SAS into a computer model.

2.5.2.2 Domino Agent

Domino is an artificial intelligence (AI) agent [24]. An AI agent is defined as an autonomous entity which can set its goals and act on the world to pursue these goals. Domino, therefore, addresses the types of cognitive processes like setting goals, generating strategies to fulfill these goals, carrying out actions to implement the strategies and reacting to environmental changes.

Domino agent provides a framework with seven types of processes operating on six types of information:

- Goal generation: based on some beliefs about its environment and current state, this process sets up a goal for the agent;
- Solution generation: this process treats the goal as a problem to be solved and finds possible strategies to fulfill the goal;
- Solution evaluation: this process compares all proposed solutions and chooses the proper one;
- Belief update: this process updates its beliefs according to the new solution;
- Plan adoption: this process determines a new plan of actions for the selected solution;
- Plan execution: this process decomposes the new adopted plan of action into atomic actions that are executed by the agent;
- Monitoring: this process monitors the effects of these actions on the world and checks how well the intended goals are achieved.

2.5.2.3 Implementation SAS in COGENT

Using buffers to store the information and rule-based processes to handle the processes, the Domino agent is ready to be implemented in COGENT. Actually, Glasspool [23]

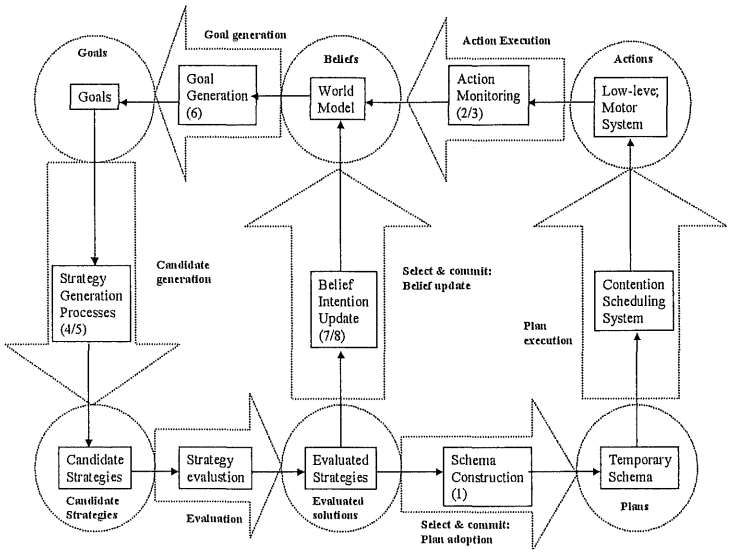


Figure 2.7: Mapping of internal processes in SAS onto Domino.

has mapped the outline of SAS [20] directly onto the Domino outline, see Figure 2.7. In Figure 2.7, the numbers in parentheses refer to the processes decomposition in the three stages mentioned in Chapter 2.3. The processes addressed in Domino are printed in boldface.

Based on this mapping, a model of SAS has been implemented in COGENT according to the Domino processes explained above [23, 25], shown as Figure 2.8. Processes are implemented in rule-based processes shown as the hexagons. Information are stored in buffers shown as the round rectangles. Arrows imply the communications between processes and buffers. CS is implemented in a compound box shown as a rectangle, in which the schema hierarchy is implemented in an interactive network.

2.6 Conclusion

This chapter reviews the literature of the current research. The theme of this research is to build a model of attention and to view how attention switches from one action to another during interruptions. Therefore, attention and two kinds of actions are

Tableau conforme à l'original.

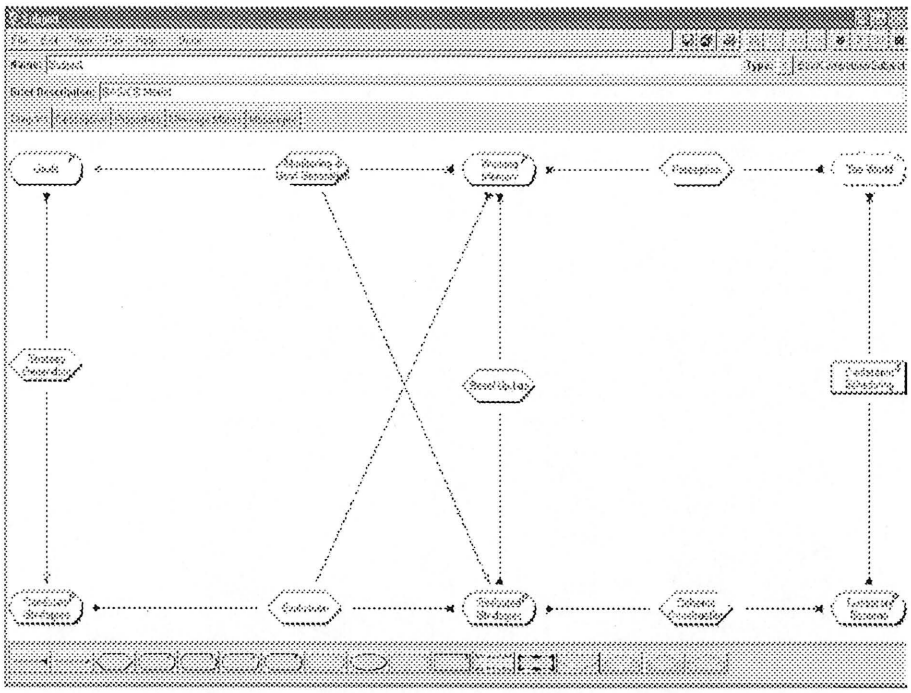


Figure 2.8: Implementation of SAS in COGENT based on Domino agent.

first explain in the chapter. With these concepts, Norman and Shallice model presents two mechanisms, the CS and SAS, which control automatic actions and willed actions respectively. The model proposes that each action is represented by a memory schema which is associated with an activation value. At any time, only the action with the highest activation is executed. The primary role of attention is in the control of action. Attention affects actions' execution by adjusting the activations of schemas.

COGENT provides a box-arrow modelling environment for cognitive objects. Different types of boxes are used to represent different components like memory buffers, cognitive processes, networks and compound boxes. Two types of arrows are used to indicate two types of communications (either read or write) between boxes. The processes are rule-based. Each rule is a if-then clause which defines a pair of condition and operation.

Under the environment of COGENT, CS has been implemented as a network interacted with object and resource. Schemas are organized hierarchically as a tree. Each schema has an activation value which is influenced by four facts: a self influence from the schema itself, a lateral influence from competitor schemas, an internal influence from high level schemas and an environmental influence from the object network and the resource network. Whenever a schema's activation value is exceed a threshold, it is selected. When a lowest-level schema is selected, it excites the proper object and resource in order to execute an action. The implementation of SAS proposes an AI agent as its approach. It specifies six processes which is implemented directly in COGENT.

But by now, there has no report about implementation of the entire Norman and Shallice model or works on the interaction between the CS and SAS. Therefore, it causes our interest on the current research. In next chapter, we will explain details about the research.

Chapter 3

Attention implementation

3.1 Introduction

This chapter presents most details of our current research. The first section is a preliminary, which includes a brief review of the theoretical and empirical background, the definition of interruption and its three cases, the objectives of the research and the methodology and its constraints. The second section concentrates on the implementing interruption in Norman and Shallice model. This part explains action representation, activation compute, interruption implementation, processes in SAS and their main functions and uses an example to describe the procedure of an interruption and the interactions between CS and SAS during the interruption. Then, the third section prints out the results of the research from two experiments, discusses the experimental results in accordance with the real world and compares the model of attention with task switching and reactive planning.

3.2 Preliminary

As discussed before, it's time to help cognitively impaired people in a technological way. To develop such technologies, first of all, we need to understand how human cognitive processes work and what factors affect these processes. Attention is one of such factors because human actions are under control of attention. Attention is what enables us to process information about the world around us. We can only be aware of things around us if we pay attention to them. Attention filters and feeds information about the world around us into our mind. This thesis aims to implement a model of attention. The model gives a view of how attention controls human actions during interruptions.

The delay of action performance leads to the goal concurrence. Goals management is necessary because two actions could not always be performed at the same time. The interruptions, which are presented later, are an example of such conflict. In our everyday lives, interruptions occur very often. For example, the phone rings as one is deeply engaged in preparing meals, or somebody knocks the door unexpectedly as one is writing a letter, or an urgent message arrives from the supervisor as one is planning a document. Each case easily creates a situation in which one's attention switches from one action to another.

3.2.1 Interruption

An interruption is defined as an unanticipated issue rising up from the environment while a main action is being performed. Facing an interruption issue, individuals mainly react in three different manners depending on the urgency.

- Case 1: if the interruption is more urgent than the current action, one switches his attention from the current action to perform the interruption action and completes the main action afterward.
- Case 2: the interruption is urgent, but the main action is currently dangerous, one still executes a part of the current action in order to keep it safe and then

accomplishes the interruption. For example, somebody knocks at the door when one is frying eggs, usually people turn off the stove then go and open the door.

- Case 3: the interruption is less urgent than the main action and will be processed later after the completion of the current action.

To deal with interruptions during action completion implies to deal with attention. An intelligent system attended to model interruptions needs to represent the action in process, the changes in the environment leading to an interruption and the decisions controlling the actions order.

3.2.2 Attention Model Implementations

Norman and Shallice, as we mentioned before, provide a psychological model composed of two mechanisms of action control, Contention Scheduling mechanism (CS) responsible for well-learned actions, and Supervisory Attentional System (SAS) required when the sequence of actions is new, when the action itself is dangerous, or when the plan must be modified according to an unanticipated environmental change [2]. Cooper and Shallice have implemented a detailed computational model of CS [19]. Shallice and Burgess proposed an outline of the processes and their interactions involved in SAS [20]. As an intelligent agent, SAS is able to plan, generate strategies and solve problems. Therefore, Glasspool used an artificial intelligence (AI) agent theory to implement the SAS [23]. But still no implementation resumes the SAS role neither the interactions between the SAS and CS, which are essential to explain how to deal with interruptions.

3.2.3 Objectives

The objectives of this thesis are to implement the entire theory of SAS and CS and the interactions between these two mechanisms regarding interruption control. Therefore, the objectives of this thesis are threefold:

- Implementing a scenario of the SAS;
- Implementing the attention switching from the SAS to the CS;
- Implementing the interruption management in the first case (see 3.2.1).

3.2.4 Methodology and Constraints

Cooper fleshed out the internal processes of SAS-CS model by using COGENT [25]. Based on this structure, the model of attention is implemented in COGENT. In this implementation, modifications are made as few as necessary for the current purpose.

As stated above, the main purpose of this thesis is to treat the Norman and Shallice's model as an entire entity and to focus on how attention switches between two control levels during an interruption. To give a simple and typical view of how attention handles an interruption, only the first case is implemented in current research. This means that the interruption management includes three steps: performing the main action; performing the interruption action as soon as it occurs; and completing the main action afterwards. Here, only one level of interruption is implemented, although sometimes new further interruptions could happen.

Regarding of the environment under which the model performs, Cooper and Shallice [19] built vivid environment for their model of CS. In their environment, resources and objects are taken into account to let the model arrange the proper resources and objects when it executes an action. For example, to pick up a spoon, the right hand (resource) and the spoon on the table (object) are required. Meanwhile, the environment can influence the schema selection of CS to some extent (depending on the amount of environmental influence). However, this research is not interested either in how the model interacts with its environment or in how much the environment could influence the action selection. Therefore, the environment here is implemented without the representation of resources and objects. By doing so, it is assumed that whenever the model needs a certain resource or object to complete an action, it is available and could be used

correctly. Moreover, since the resource is ignored here, the model of attention cannot simulate two actions synchronously and cannot be used to simulate process errors caused by interference of requests of a resource or an object. It performs actions separately. In addition, in current implementation, the environmental influence is ignored as the result of simplification of the environment implementation.

3.3 Implementation of Interruption in Norman and Shallice Model

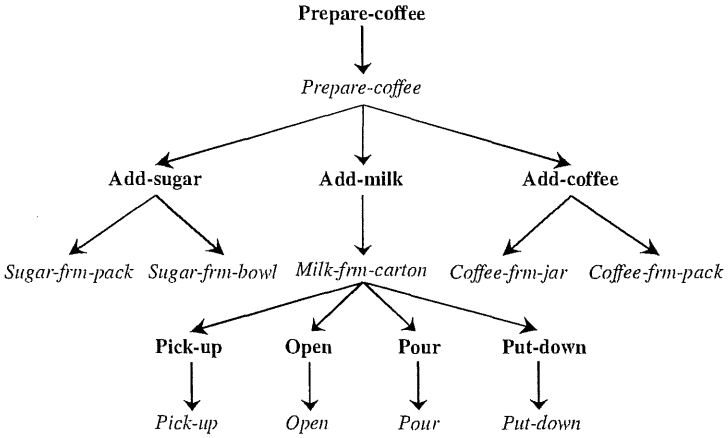
In this chapter, the interactions between CS and SAS are implemented to show how attention switches from one action to another during an interruption. Section 3.3.1 describes two actions representation in the current implementation. The following two sections 3.3.2 and 3.3.3 talk about the implementation of the two mechanisms of action control, CS and SAS. Section 3.3.5 discusses the interactions between CS and SAS during an interruption. And the last section (3.3.6) presents how interruption is implemented for the current purpose.

3.3.1 Action Representation

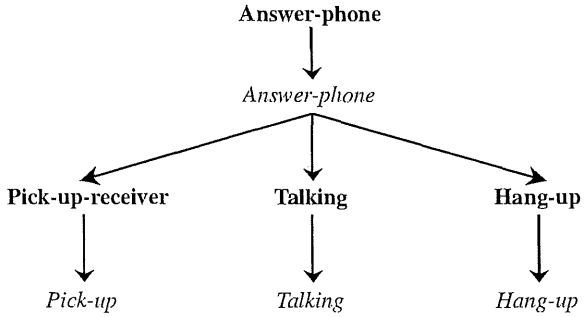
3.3.1.1 Hierarchy Representation

Proposed by Cooper and Shallice in their model of CS [19], schemas are goal directed. Thus an action is represented in a hierarchy within which schema levels alternate with goal levels. In the current implementation two actions are taken into account: the prepare-coffee action and the answer-phone action. Figure 3.1 presents their schema/goal organizations respectively. The leaves of the schema/goal representation correspond to the atomic actions which could be executed directly in the world.

Each schema is associated with an activation value. The activation value is a real number and varies over time. Section 3.3.6 discusses the details about how a schema's



(a) Schema/goal organized for the action of coffee preparation.



(b) Schema/goal organized for the action of answer phone.

Figure 3.1: The trees relate to schemas and goals. Goals are printed in bold type and schema in italic type.

activation value updates during the model's performance.

3.3.1.2 Urgent Priority Value

As emphasized before, the theme of this research is to model attention controlling actions during interruption. Therefore an additional parameter is specified in our implementation. It is called urgent priority value. This value identifies how urgently an action has to be executed. Thus, an action is represented as a pair of its name with its priority value like **Action (Name, Priority)**. **Name** identifies the action's schema hierarchy as any one of the trees in Figure 3.1. **Priority** is the action's urgent priority value. At any

time, if an action's urgent priority value is higher than any others, it should be executed first.

During the simulation of the model of attention the prepare-coffee action is initialized as the main action which is interrupted by the interruption: answer-phone action. Therefore, the prepare-coffee action is assigned a lower priority value than that of answer-phone action for current purpose. Section 3.3.6 will give more details on this aspect.

3.3.2 CS Implementation

As mentioned before, our objective is to implement interruption in Norman and Shallice model and by doing so to give a view of how attention switches between actions during an interruption. Therefore, in current implementation, we remove the Object Network and Resource Network from the former implementation of CS [19]. We assume that at any time when the subject needs allocate some objects or resources to take out an action, they are available and can be used properly. Thus, the current implementation of CS is composed of the schema hierarchy and the selection mechanism.

3.3.2.1 Schema Hierarchy

The core of the CS is the schema hierarchy, within which each schema represents an action [26]. Two sub-nets, named *coffee* and *phone* respectively, are defined in the *Schema Hierarchy* to refer to the two actions in the current implementation. Each schema must be identified by its description and the sub-net it belongs to. This is because an action is often involved to contribute to different goals. For example, the schema *milk-from-carton* could be used for a preparing coffee goal, or for a baking cookies goal, or even for a frying eggs goal.

Two processes, the *trigger schemas* and the *select & act*, operate on the schema hierarchy. First, the *trigger schemas* updates each schema's activation value. Then, the *select & act* works as the schema selection mechanism to select the proper schema and leads an action in the world when a lowest-level schema is selected.

3.3.2.2 Activation Value

Methodology

COGENT provides a set of parameters to contribute the activation value calculation. These parameters are binding on the *Schema Hierarchy* network. The **Min Act parameter** and the **Max Act parameter**are set to ensure schema activation varies over time within a range of [0.0, 1.0]. The **Lateral parameter** and **Self parameter** are set through some trials according to the following criterions: 1) each schema, even the lowest-level schema, has to get an activation value which allows it to be selected; 2) both the lateral influence and the self influence should belong in a comparable rang.

Schema Activation Updating

Schema activation value is updated on each cycle during the model's performance [26]. The process *trigger schemas* calculates each schema's activation value on each cycle. The activation update function is expressed as equation (3.1) [19]:

$$A_i^{t+1} = \overline{\sigma}[P\overline{\sigma}^{-1}(A_i^t) + N_i], \tag{3.1}$$

where P is the network's persistence parameter which could be any real number. It describes the degree to which activation values persist in the absence of excitation. For example, a persistence of 1.0 will lead to nodes maintaining their current activation in the absence of further excitation. While a persistence of 0.0 will lead to nodes reverting immediately to rest activation in the absence of excitation. A_i^t, which belongs to the interval 0 and 1, is node i's activation on cycle t; N_i is the net input to node i; and $\overline{\sigma}$ is the standard sigmoid activation function (σ) shifted and scaled such that:

$$\begin{aligned} \overline{\sigma}(\infty) &= 1.0 \\ \overline{\sigma}(0) &= 0.1 \\ \overline{\sigma}(-\infty) &= 0.0 \end{aligned}$$

The net input N_i of a schema is determined by four factors: internal influence, environmental influence, self influence and lateral influence.

Internal Influence

The internal influence, also called the top-down influence which comes from the parent schema or from SAS directly, is calculated as follows (equation (3.2)).

Let I_i be the internal influence on a schema i, let A_p be the activation value of its parent and let N_p be the number of the children of that parent p,

$$I_i = \mathbf{A_p}/\mathbf{N_p}. \tag{3.2}$$

When the schema is directly triggered by SAS, the internal influence from SAS on that schema is set to 1.0.

Environmental Influence

The environmental influence comes from the object and the resource with which the model interacts during action performances [19]. For example, a bowl of sugar and a free hand could give an excitation on the schema *sugar-from-bowl*. In current implementation, the environment is implemented without resource network and object network. The object network and resource network are removed from CS by assuming that at any time when the subject needs allocate some objects or resources to take out an action, they are available and can be used properly. Therefore, the environmental influence is ignored here.

Lateral Influence

Competitive schemas inhibit each other. This inhibition is called a lateral influence. Competitive schemas are defined as those sharing either the same goal or one or more subgoals in common. For example, in Figure 3.1(a), schema *sugar-from-bowl* is a competitor of the schema *sugar-from-pack*, because they share the same goal **Add-sugar**. At any cycle t, the lateral influence on a schema is the sum of the influences from all its competitors, which is expressed as equation (3.3):

$$L_i = \sum_k L_p(A_k^t - L_b) \tag{3.3}$$

where A_k^t is the current activation of any competitor of that schema; and two parameters, L_p and L_b, both of which could be any real numbers, control how lateral influences from competitor schemas are calculated.

Self Influence

A schema is influencing on its own activation. It is called self influence, which is proportional to the schema's activation value. Equation (3.4) explains how self influence on a schema i is calculated:

$$S_i = S_p(A_i^t - S_b) \tag{3.4}$$

where A_i^t is the schema's activation on current cycle i; and two parameters S_p and S_b, both of which could be any real numbers, control how the schema's activation contributes to its self influence.

Finally the net input to schema i, N_i is calculated as equation (3.5):

$$N_i = [(C/I_i)C + (1 - C/I_i)I_i] \tag{3.5}$$

where I_i is the internal influence on that schema; C is a parameter, which represents the total competitive input to the schema and is scaled by the schema's self influence and lateral influence, shown as equation (3.6):

$$C = [(S_i/L_i)S_i + (1 - S_i/L_i)L_i] \tag{3.6}$$

where S_i and L_i are respectively the self influence and lateral influence on the schema i.

3.3.2.3 Selection Mechanism

The process *Select & Act* in CS works as the selection mechanism. It is in charge of selection of the most proper schema and carries out an action when a lowest-level schema is selected. Whenever a schema's activation value exceeds a threshold, which is

Table 3.1: The main processes and their functions in SAS.

SAS Component	Process Function
Perception	Senses any environmental change and transfer useful information to the subject.
Monitoring & Goal Generation	Monitors action execution and sets goals if necessary.
Strategy Generation	Generates different strategies to achieve the set goal.
Evaluation	Evaluates the strategies and selects the most suitable one currently.
Schemas Construction	Constructs a temporary new schema for the select strategy.
Beliefs Update	Updates beliefs about the environment.

set to 0.60, and is higher than any of its competitor's activation values, it is selected. A competitor of a schema is a schema that may share the same goal or share one or more subgoals in common. For example, in Figure 3.1, the schema *sugar-from-bowl* is a competitor of the schema *sugar-from-pack*, because they share the same goal **Add-sugar**. A selected source schema in turn excites its component schemas. After achieving its goal, a schema should be inhibited (sending a negative exciting value) and be deselected.

3.3.3 SAS Implementation

We continue to use the structure fleshed out by Cooper [23, 25]. Modifications are made for the current purpose. Table 3.1 lists the processes involved in the structure and their major functions during action control. Among these processes, we highlight two in the following example: *Perception* and *Monitoring & Goal Generation*. These play the most important role during the response to interruptions.

As mentioned above, all processes are implemented in rule-base processes in COGENT. That implies that each of the processes is based on rules. A rule usually is a if-then pair. Operations could be any COGENT defined or user defined functions. The

basic rules of each process in the current implementation are described as below.

Perception

This rule detects a new action from the world and adds it into Working Memory.

```
{If action (T, U) in the World
    not do-action (T, U) in Working Memory
 Then add do-action (T, U) in Working Memory.}
```

Monitoring & Goal Generation

Rule 1 initials an action and sets a goal for it.

```
{If do-action (T, U) in working memory
    not current-strategy (T, S) in Working Memory
 Then add strategy-generate (T) in Goals}
```

Rule 2 monitors the interruption while the subject is doing the main action. It makes decision of attention switching by calling a function, which is discussed in next section.

```
{If do-action (Ti, Ui) in working memory
    not current-strategy (Ti, Si) in Working Memory
    is-doing (Tc, Uc) in Working Memory
Then
    call Compare (Ui, Uc) }
```

Strategy Generation

This rule is used to generate strategies for a new goal.

```
{If strategy-generate (T) in Goals
 Then add (Strategy-T) in Candidate Strategy}
```

Evaluation

This rule evaluates candidate strategies. In the current implementation, we only implemented one strategy for both of the actions' goal. Therefore, the strategy is selected directly.

```
{If (Candidate) in Candidate Strategy
 Then add selected(Candidate) in Evaluated Strategies}
```

Schemas Construction

This rule constructs a new temporary schema for the action that the subject is going to do next. It also deletes all the outdated temporary schemas.

```
{If selected(S) in Evaluated Strategies
    not schema(s) in Temporary Schema
 Then delete all schema(X) from Temporary Schema
      add schema(s) in Temporary Schema}
```

Beliefs Update

This rule updates the belief about the current strategy.

```
{If selected(S) in Evaluated Strategies
    do-task(T) in Working Memory
    not current-strategy (T, S) in Working Memory
 Then delete all current-strategy (X,Y) from Working Memory
      add current-strategy (T, S) in Working Memory}
```

3.3.4 Example

Initiation

In this implementation, it is assumed that at the initial time, the subject wants to prepare a coffee. The first process in SAS, *Perception*, adds a request of the coffee preparation action in the working memory. Then *Monitoring & Goal Generation* sets up the goal of coffee preparation. The *Strategy Generation* in turn generates a possible approach to achieve the goal. Currently, to make a coffee by using instant coffee is the one and only strategy for the goal. Therefore, after evaluation, this strategy then is selected to construct a temporary new schema for coffee preparation. The temporary new schema triggers the source schema of coffee preparation action in the CS. So far, SAS delegates control of action to CS.

Perception of Interruption

Now the coffee preparation action is controlled by the CS unless an interruption occurs. At this time *Perception* acts as an "advisor" to indicate the change in the environment. When *Perception* perceives the phone ringing, it feeds the request to answer the phone.

Attention Switching Determination

Then a central function Compare (Ui, Uc) in *Monitoring & Goal Generation* is responding to the determination of attention switching during interruptions. Let **Action (C, Uc)** and **Action (I, Ui)** represent the current action and the action for the interruption respectively, the function is described as following:

```
{If Ui > Uc
 Then set a goal for I in Goals;
        marker (stop(C)) in working memory;
        {If finished (I)
         Then resume (C)}
 Else {If finished(C)
       Then set a goal for I in Goals} }
```

The function in *Monitoring & Goal Generation* first will compare the new interruption action's urgent priority with that of the current action. If it is higher, another rule in the process will be fired to raise a goal for the new action and pause the current action at the same time. This goal will lead to another flow of processes in SAS and reconfigures CS to adopt another schema hierarchy for the action of answering the phone. After finishing the phone call, the cue of an unfinished action of coffee preparation reminds the subject to resume it. Here, the unsatisfied goal is retrieved from the working memory. A goal then should be set again to continue the former coffee preparation action. The processes flow of SAS will finally reconfigure CS to handle it until completion unless another interruption occurs. Cues in the world help the subject to determine where to continue the former action.

3.3.5 Interactions between CS and SAS

In Chapter 2, it is described that SAS takes over control of actions through vertical threads. In the current implementation, SAS interacts on CS in three main spots. That means three vertical threads are involved in the implementation.

First, at the initial stage, SAS initializes a routine action and delegates it to CS to be completed automatically if no interruption occurs.

Second, when an interruption takes place during the routine action performance, SAS switches attention from the routine action to the interruption by constructing and implementing a new temporary schema for the interruption action as well as pausing the initial routine action.

Third, when the action of interruption is done, SAS updates beliefs about its situation and reconfigures CS to continue the unfinished former routine action until completion if no other interruptions occur.

3.3.6 Interruption Implementation

Only the first case of interruptions is implemented in this research. Therefore, whenever the subject perceives that the phone is ringing (through the *Perception* process), it switches its attention from the current coffee preparation to answering the phone. For this purpose, the answer phone action is assigned to an urgent priority of 30 while the coffee preparation to 10.

The model interacts with its world, which is called *Experimental World*. The phone call interruption occurs randomly from the world. In *Experimental World*, a process *Manipulate World* responds to the interruptions' random. As in many other computer programming language, the rule-based language of COGENT offers the random number to contribute to this issue. In this implementation, a random number R is introduced. Whenever R is numerically equal to 5, a request of answer phone is added in the *State of the world*.

3.4 Results and Discussions

In this Chapter, the results of two experiments are reported to explain how the model of attention works. Results of these two simulations are presented in groups of cycles. Then the experimental results are explained through their accordance with the real world. Afterwards discussions are divided into three parts: 1) the model of attention is reviewed and some ongoing works are proposed; 2) the model of attention is compared with another task switching model built by Sohn and Anderson in ACT-R [27]; and 3) a parallel is drawn between the model of attention and AI concepts.

3.4.1 Results

In this section, results of the simulation are reported based on two experiments, both done in COGENT. As mentioned before, the time when the phone call interruption

Tableau conforme à l'original.

occurs is set randomly. In the following two experiments, the interruption feeds in the model on different cycles. Results are presented in groups of cycles to show how attention controls actions. Some representational cycles are listed and explained afterwards. On each cycle, the information transference can be obtained from the message page of any activated windows. More visually, COGENT provides the Activation Graph of the Schema Hierarchy to give a dynamic view of nodes' activation value, which updates over time. Figure 3.2 provides a view of the schemas' activations when the subject is answering the phone. As mentioned previously, while answering the phone, the initial action of coffee preparation is paused. Therefore, in Figure 3.2, the schemas belonging to the sub-net of *coffee* are inhibited.

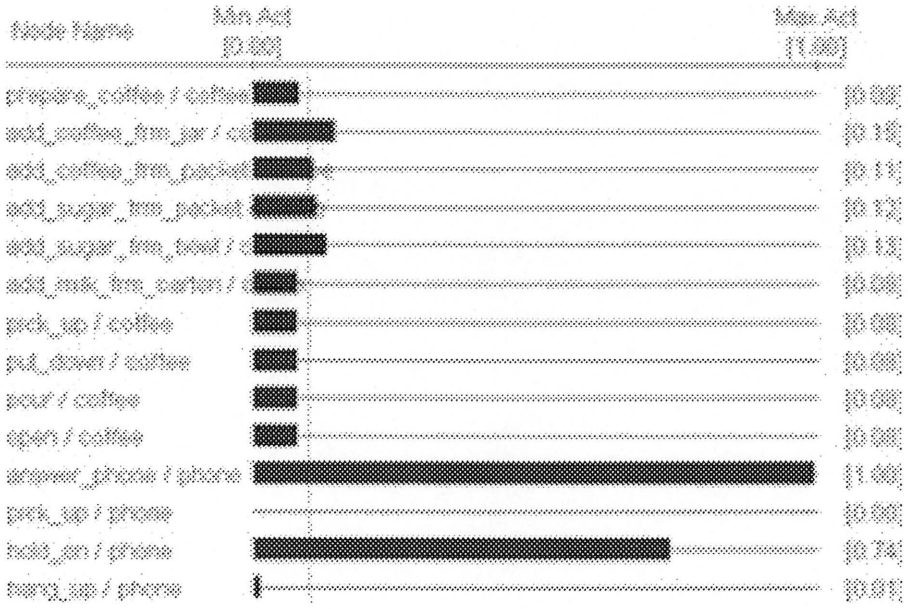


Figure 3.2: The Activation Graph histogram.

3.4.1.1 Experiment 1

In the first experiment, the answer-phone action feeds in at cycle 33. As we stated earlier, during the execution of the model, the traffic between components could be viewed

through messages of each component's window. Through these message views, we get the information of each cycle. Some of them are highlighted below.

- From cycle 1 to cycle 16, preparing the action of coffee preparation. During this time duration, SAS initializes the main action;
- From cycle 16 on, SAS delegates control of action to CS. The current action is coffee preparation and schemas' activations are updating in CS;
- From cycle 33 to cycle 50, preparing to answer the phone. During this time duration, information flows within SAS to switch attention to the action of answering phone call, while CS continues with the main action of coffee preparation;
- At cycle 41, schema *add-coffee-from-packet/coffee* is selected;
- At cycle 49, schema *add-sugar-from-packet/coffee* is selected;
- From cycle 50 to cycle 123, executing the action of answering the phone. Now schemas belonging to answer phone are excited enough by SAS and are executed by CS automatically;
- From cycle 124 to cycle 139, resuming the former action of coffee preparation. During this time, SAS returns attention to the interrupted action of coffee preparation;
- From cycle 139 to the end of the model, executing and finishing the action of coffee preparation.

3.4.1.2 Experiment 2

In the second experiment, the answer phone action feeds in at cycle 24. Messages of this experiment are listed below.

- From cycle 1 to cycle 22, SAS initializes the action of coffee preparation;

- From cycle 22 on, SAS delegates control of action to CS. The current action is coffee preparation;
- From cycle 24 to cycle 36, SAS is preparing to answer the phone. Meanwhile the coffee preparation is paused;
- From cycle 37 to cycle 123, SAS delegates control of action to CS to execute and complete the action of answering phone;
- From cycle 124 to cycle 135, SAS is resuming the former action of coffee preparation;
- From cycle 135 on, SAS delegates control of action to CS, in which the schemas' activation is excited again;
- At cycle 157, the schema *add-sugar-from-bowl/coffee* is selected;
- At cycle 163, the schema *add-coffee-from-packet/coffee* is selected;
- At cycle 182, the schema *add-milk-from-carton/coffee* is selected;
- From cycle 183 to the end of the model, the action of coffee preparation is completed with no further interruption.

Whenever attention is required, SAS interferes in the performance of actions. As discussed before, SAS put attention to an action through modifying its corresponding schema's activation. Therefore, if a schema is triggered directly by SAS, it gets an excitation of the amount of 1.0 from SAS.

The CS, as its name implies, is in charge of two facets of schemas' mechanism: contention and scheduling. Firstly, competitor schemas compete with each other. For example, from the two experiments, we see that the schema selected to achieve a certain goal is not always the same. In Experiment 1, the schema *add-sugar-from-packet/coffee* is selected to achieve the goal of add sugar; while in Experiment 2, the schema selected for this goal is the schema *add-sugar-from-bowl/coffee*.

Secondly, as discussed in section 5.1.1, schemas are goal directed. In addition, a goal could be decomposed into a set of subgoals. These subgoals are either equally ordered or scheduled in a sequence. For example, the order of the schema's selection is different in the two experiments. In Experiment 1, the schema *add-coffee-from-packet/coffee* is selected before the schema *add-sugar-from-packet/coffee*. In Experiment 2, the schema *add-sugar-from-bowl/coffee* is selected before schema the *add-coffee-from-packet/coffee*. Because these two schemas are equally ordered, the order of their selection is a result of competition.

Notice that in Experiment 1, the phone call feeds in on *cycle* 33 and the action of answering the phone begins at *cycle* 50. In this duration of its excitement, the activations of some schemas in coffee preparation are still increasing. This could be explained in the real world: when the phone rings while we are executing another routine action, most times, we need a period of time to respond to the ringing phone. Until we pick up the receiver, we continue our performance of the former action, though it may be inhibited unintentionally at the very second that the phone rings.

In the implementation, SAS interacts with CS on several stages. First of all, SAS initializes a routine action (e.g., coffee preparation) and delegates it to CS to complete by executing a set of actions. Second, when an interruption (e.g., phone ringing) occurs during the routine action performance, SAS switches attention from the former action to the interruption by constructing and implementing a new temporary schema for the interruption action, as well as pausing the former action. Third, when the action of interruption is done, SAS updates beliefs about its situation and reconfigures CS to continue the former routine action to completion if no other interruptions occur.

3.4.2 Discussions

3.4.2.1 Review of the model

The model of attention gives a view of the role of attention in action controlling through coping with interruptions. With the introduced parameter of the urgent priority,

the model could recognize an interruption and make a decision about the direction to which the attention should be driven. SAS oversees the execution of action and the state of the environment. Whenever necessary, it interferes with the action control and finally delegates control of action to CS. Schemas in CS are organized hierarchically, which incarnate the integrality and conciseness of the model. Because the model here treats the CS and SAS as a whole, it did not put more ink on either of the two. Parameters are settled on through several experiments until the model performs properly. Interruptions are simulated randomly like most situations in the real world. The model can react and handle these random interruptions successfully as expected. However, considering the model in accordance with the real world, it could be improved, at least in some aspects.

First, in current implementation, the urgent priority is assigned to be a constant. But in the real world, depending on its different processing stages, how urgent an action is may vary over time. For example, the coffee preparation action is less urgent at the very beginning. But while the stove is turned on to boil the water, its degree of urgency should increase according to the current danger of fire. Therefore, a more mature analysis should take into account the variation of this parameter according to time and to the hazardous status of the action.

Second, to simplify the routine actions execution, the experimental world is implemented without the representation of objects and resources. Therefore in our simulations, the model performs actions separately. That means, for example, when a phone call comes during the coffee preparation procedures, it pauses the coffee preparation action to answer the phone call, then goes back to preparing coffee after conversation. However, in many situations, people can do these two actions together at the same time. In fact, in daily life, people often do more than one things simultaneously. Processing resources should be taken into account to let the model simulate synchronous performance of two actions. On the other hand, because of the ignored objects and resource implementation, the model here did not simulate process errors caused by interference of requests of a resource or an object.

Another issue that takes place often in our daily life is that people's actions are not

always interrupted by a single interruption. That means a complex action, because of its relatively long time duration and deep decomposition of stages, may be disturbed by a series of interruptions. For example, while a clerk is preparing his weekly work plan, the phone rings; then while he is answering the phone, his manager asks him to attend an emergency meeting. The model can simulate human cognition more effectively if this aspect is taken into account.

3.4.2.2 Comparison with Task Switching in ACT-R

As discussed before, switching from one action to another is under the control of attention. The model we implemented focus on the role of two mechanisms, CS and SAS, during attention switching from one action to another. To carry out any action could be treated as a task. Therefore, attention switching between actions is often referred using the term of "task switching". Many studies are dedicated to task switching. An example is provided by M. H. Sohn and J. R. Anderson with their model of task switching in ACT-R [27].

As discussed in 1.5.2, the ACT-R system is a general system for modeling higher level cognition. It could be characterized as both symbolic and subsymbolic. At the symbolic level, information processing in ACT-R involves a sequence of production rule firings. Each production rule involves some chunks' retrieval to transform the goal status [14,27]. At the subsymbolic level, ACT-R assumes that multiple production rules compete to determine the schedule of information processing. In addition, the retrieval of a chunk depends on its activation value. This idea is similar to the CS mechanism. Therefore, ACT-R model could be used to simulate executive and automatic actions. In fact, in their model, Sohn and Anderson investigated how executive and automatic control mechanisms have their effects when people rapidly switch from one task to another [27].

But there was no SAS in the ACT-R model. In ACT-R, the calls for shift of attention are controlled by explicit firing of production rules [28]. However, changes in the environment could be transferred into declarative chunks which can then be processed

by the ACT-R system. When its goal is changed (which means its task is switched to another), the model retrieves some other chunks according to the new goal and determines the next task. Therefore, attention switches automatically from one task to another.

In a word, the key difference between the two models is: in the SAS-CS model, attention is directed by SAS while in Sohn and Anderson's model, switching of attention is controlled by production rules. In addition, Sohn and Anderson's model is focused on how executive and automatic control mechanisms have their effects when people rapidly switch from one task to another. In contrast, the model here focuses on how attention could control actions through interaction of the SAS and CS.

3.4.2.3 Comparison with Reactive Planning

In this research, the model interacts with an experimental world, from where an interruption takes place. Dealing with context changes is often referred to as another AI concept: reactive planning. Planning is a key ability for intelligent systems, increasing their autonomy and flexibility through the construction of sequences of actions to achieve their goals. More precisely, reactive planning extends the traditional planning by adding the current context influences in the action selection [29]. Action selection is supported by reactive plans, which are stored structures specifying the next act according to both current internal and environmental contexts. However, reactive planning chooses only the immediate next action. A deliberate planning may inform or create new plans or behaviors. Therefore a complete intelligent agent combines three-layers [30]:

1. Behavior-based AI, the decomposition of intelligence into simple, robust, reliable modules;

2. Reactive planning, the ordering of expressed actions via carefully specified program structures;

3. Deliberative planning, which may inform or create new plans or behaviors.

The SAS-CS model presented here are parallel to such an agent. The first layer is at the motor level and is part of the CS; the second layer is the CS itself; and the third layer is the deliberate level and corresponds to the SAS.

In psychology, attention is responsible for context awareness. In the model of attention, one of the SAS' roles is to detect environmental changes. The model of attention in this paper presents SAS directing attention onto a specified individual schema in CS to control action. Therefore, the automatic process is then isolated from the decisions requiring much more elaborations. It could be compared with the Basic Reactive Plan, which is explained by Bryson and Stein [29]. Both of the models select actions based on priority levels and hierarchical plans. But, the choice of the next action is embodied in the action representation in the Basic Reactive Plan. In the current model, action selections are divided into two cases: for automatic actions, action selection is done by CS based on schemas' activation value; for novel actions, the next action is chosen by other SAS processes [26]. Two SAS processes which are involved in case of interruption are: the *Perception* and *Monitoring & Goal Generation*. These two processes respectively respond to the interruption identification and the action selection based on the priority of urgency.

The model presented in this paper is not intended to present the whole mechanism of the Norman & Shallice model. It focuses on how attention switches and handles actions when confronting an interruption. First, it shows how an environmental change leads to an interruption in the CS process and second, how the SAS gives back the control to the CS when the chosen action becomes automatic.

3.5 Conclusion

This chapter describes the model of attention, which is the theme of this thesis. The model is based on the Norman and Shallice model, therefore, during the implementation we continue using the approach discussed in section 2.5 for current purpose. An interruption is defined as an unexpected action that need to be handled immediately while one is

doing another action. To identify the two actions, an additional parameter called urgent priority is bound to each action. The interruption always has a higher priority than the main action does in order to simulate the interruption. Activation compute is explained with a set of parameters offered by COGENT. The implementation of SAS focuses on two important processes: Perception and Monitoring & Goal generation. The function of each process is described through a set of rules. The logics of the main rules of each process are presented. Next, results from two simulations of interruption are listed with program executing cycles. Finally, the whole model is discussed through comparisons with two other researches: Task Switching in ACT-R and Reactive Planning.

Conclusion

In this thesis, a cognitive model is presented to examine how attention switches in order to control actions during interruptions. Concepts of action, attention and interruption are described with examples. Everyday, people perform many actions. These actions are driven by particular goals even if these goals are not always conscious. Actions requiring less awareness are called well-learned behaviors or automatic actions. Other kinds of actions need much more attentions to be executed successfully. These actions are called willed behavior. Responding to these two kinds of actions, Norman and Shallice's model addressed two distinct mechanisms of control, CS and SAS. They work together as an intelligent agent that is able to raise its goal according to beliefs about its current environment, to generate possible strategies for problem solving, to evaluate these strategies and to select the most proper one, allocate relative resources to carry out actions, and update its beliefs about the environment as well.

Taking both CS and SAS into account, the model is implemented in COGENT. The SAS-CS models the way humans perform actions under attention control. The simulation presented here highlights the interactions between these two agents and how interruption stops a current action. It sketches the link between the two components of the model, SAS and CS. SAS plays a referee role between the current action and the interruption action. Then, SAS gives back the control to CS when action schemas become automatic. The model proposed successfully simulates the real world. Some adjustments are to be made to take into account other cases of interruption, a series of interruptions and the evolution of the urgent priority.

Appendix

World State

Class: Box/Buffer/Propositional

Access: Random

Initial Contents:

Element: *Initial task*

```
task(prepare_coffee, 10)
```

Current Contents (of World State):

0 : task(prepare_coffee, 10)

Manipulate World

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (refracted): *Random Interruption*

IF: not `is_doing(answer_phone)` is in *World State*

`R` is a random integer drawn from `0` to `100` inclusive

`R` is numerically equal to `5`

THEN: add `task(answer_phone, 30)` to *World State*

Rule 2 (refracted): *Delete a task after its completion*

IF: `task(T, U)` is in *World State*

`finish_action(T)` is in *World State*

THEN: delete `task(T, U)` from *World State*

Rule 3 (refracted): *Stop the model after all tasks are finished*

IF: not `task(T, U)` is in *World State*

THEN: send `stop` to *Attention during an interruption*

Perception

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (refracted): *Add a new task into Working Memory*

IF: `task(T, U)` is in *Experimental World:World State*

not `do_task(T, U)` is in *Working Memory*

THEN: add `do_task(T, U)` to *Working Memory*

Rule 2 (unrefracted): *Delete an old belief about a task*

IF: `do_task(T, U)` is in *Working Memory*

not `task(T, U)` is in *Experimental World:World State*

THEN: delete `do_task(T, U)` from *Working Memory*

Rule 3 (refracted): *Add current action into Working Memory*

IF: `is_doing(Action)` is in *Experimental World:World State*

not `is_doing(Action)` is in *Working Memory*

THEN: add `is_doing(Action)` to *Working Memory*

Rule 4 (refracted): *Add a completed action into Working Memory*

IF: `finish_action(T)` is in *Experimental World:World State*

not `finish_action(T)` is in *Working Memory*

THEN: add `finish_action(T)` to *Working Memory*

Rule 5 (refracted): *Continue coffee task after finishing interruption*

IF: `finish_action(answer_phone)` is in *Experimental World:World State*

`task(prepare_coffee, 10)` is in *Experimental World:World State*

`pause(prepare_coffee)` is in *Working Memory*

THEN: add `continue_task(prepare_coffee, 10)` to *Working Memory*

Monitoring & Goal Generation

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (refracted): *Trigger generation of a new strategy to handle a request*

IF: `do_task(T, U)` is in *Working Memory*

not `current_strategy(T, Strategy)` is in *Working Memory*

THEN: add `generate_strategy(T)` to *Goals*

Rule 2 (refracted): *Trigger generation of a new strategy to handle a request*

IF: `continue_task(T, U)` is in *Working Memory*

not `current_strategy(T, Strategy)` is in *Working Memory*

THEN: add `generate_strategy(T)` to *Goals*

Rule 3 (refracted): *Interruption decision*

IF: `do_task(T1, U1)` is in *Working Memory*

`is_doing(T1)` is in *Working Memory*

`do_task(T2, U2)` is in *Working Memory*

`U2` is greater than `U1`

THEN: add `pause(T1)` to *Working Memory*

delete all `selected(T)` from *Evaluated Strategies*

Strategy Generation

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted):

IF: `generate_strategy(prepare_coffee)` is in *Goals*

THEN: add `prepare_coffee` to *Candidate Strategies*

Rule 2 (unrefracted):

IF: `generate_strategy(answer_phone)` is in *Goals*

THEN: add `answer_phone` to *Candidate Strategies*

Evaluation

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted): *Rule*

IF: `Candidate` is in *Candidate Strategies*

THEN: delete all `selected(S)` from *Evaluated Strategies*

add `selected(Candidate)` to *Evaluated Strategies*

Schema Construction

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted): *Create a temporary schema for the selected strategy*

IF: `selected(S)` is in *Evaluated Strategies*

not `schema(S)` is in *Temporary Schema*

THEN: delete all `schema(X)` from *Temporary Schema*

add `schema(S)` to *Temporary Schema*

Rule 2 (unrefracted): *Delete temporary schemas for non-selected strategies*

IF: `schema(S)` is in *Temporary Schema*

not `selected(S)` is in *Evaluated Strategies*

THEN: delete `schema(S)` from *Temporary Schema*

Belief Update

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted): *Update beliefs with new strategy*

IF: `selected(T)` is in *Evaluated Strategies*

`do_task(T, U)` is in *Working Memory*

not `current_strategy(T, T)` is in *Working Memory*

THEN: delete all `current_strategy(_, _)` from *Working Memory*

add `current_strategy(T, T)` to *Working Memory*

Rule 2 (unrefracted): *Rule*

IF: not `do_task(T, U)` is in *Working Memory*

`current_strategy(T, Strategy)` is in *Working Memory*

THEN: delete `current_strategy(T, Strategy)` from *Working Memory*

Schema Hierarchy

Class: Box/Network/Interactive

Properties (free):

Initialise: 'Each Trial'	**Min Act:** 0.00	**Max Act:** 1.00
Rest Act: 0.10	**Update Function:** 'CS'	**Persistence:** 0.90
Noise: 0.00	**Initial Acts:** uniform	**Act Parameter A:** 0.05
Act Parameter B: 0.15	**Lateral Influence:** 'Sub Net'	**Lateral Function:** 'Sum'
Lateral Baseline: 'Rest Act'	**Lateral Parameter:** 0.15	**Self Influence:** 1
Self Baseline: 'Rest Act'	**Self Parameter:** 0.50	

Initial nodes and Activation Graph at cycle 0:

Input Nodes	Min Act	Max Act
	[0.00]	[1.00]
prepare_coffee / coffee:		[0.06]
sugar_from_pack / coffee:		[0.06]
sugar_from_bowl / coffee:		[0.08]
coffee_from_pack / coffee:		[0.12]
coffee_from_jar / coffee:		[0.06]
milk_from_carton / coffee:		[0.06]
open / coffee:		[0.08]
pour / coffee:		[0.12]
pick_up / coffee:		[0.13]
put_down / coffee:		[0.06]
answer_phone / phone:		[0.07]
pick_up / phone:		[0.12]
hold_on / phone:		[0.10]
hang_up / phone:		[0.11]

Schema States

Class: Box/Buffer/Propositional

Access: Random

Description (of Schema States):

Hold schema-goal relation

Initial Contents:

Comment: *for coffee preparation*

Comment: *Level 3:* ————————————————————————————

Element: *The coffee_schema's goal:*

`schema_goal(prepare_coffee / coffee, prepare_coffee)`

Element: *The coffee_schema's subgoals:*

`schema_subgoals(prepare_coffee / coffee, [add_sugar, add_coffee, add_milk])`

Element: *The coffee_schema's order constraint:*

`ordering_constraint(prepare_coffee / coffee, add_coffee < add_milk)`

Element: *The coffee_schema's order constraint:*

`ordering_constraint(prepare_coffee / coffee, add_sugar < add_milk)`

Element: *The coffee_schema's initial selection status:*

`schema_status(prepare_coffee / coffee, unselected)`

Comment: *Level 2:* ————————————————————————————

Element: *The sugar_from_pack schema's goal:*

`schema_goal(sugar_from_pack / coffee, add_sugar)`

Element: *Nothing*

`schema_subgoals(sugar_from_pack / coffee, [])`

Element: *The sugar_from_pack schema's initial selection status:*

`schema_status(sugar_from_pack / coffee, unselected)`

Element: *The sugar_from_bowl schema's goal:*

`schema_goal(sugar_from_bowl / coffee, add_sugar)`

Element: *Nothing*

`schema_subgoals(sugar_from_bowl / coffee, [])`

Element: *The sugar_from_bowl schema's initial selection status:*

`schema_status(sugar_from_bowl / coffee, unselected)`

Element: *The coffee_from_pack schema's goal:*

`schema_goal(coffee_from_pack / coffee, add_coffee)`

Element: *Nothing*

`schema_subgoals(coffee_from_pack / coffee, [])`

Element: *The coffee_from_pack schema's initial selection status:*

`schema_status(coffee_from_pack / coffee, unselected)`

Element: *The coffee_from_jar schema's goal:*

`schema_goal(coffee_from_jar / coffee, add_coffee)`

Element: *The coffee_from_jar schema's initial selection status:*

`schema_status(coffee_from_jar / coffee, unselected)`

Element: *Nothing*

`schema_subgoals(coffee_from_jar / coffee, [])`

Element: *The milk_from_carton schema's goal:*

`schema_goal(milk_from_carton / coffee, add_milk)`

Element: *The milk_from_carton schema's subgoals:*

`schema_subgoals(milk_from_carton / coffee, [pick_up, open, pour, put_down])`

Element: *The milk_from_carton schema's order constraint:*

`ordering_constraint(milk_from_carton / coffee, open < pour)`

Element: *The milk_from_carton schema's order constraint:*

`ordering_constraint(milk_from_carton / coffee, pour < put_down)`

Element: *The milk_from_carton schema's initial selection status:*

`schema_status(milk_from_carton / coffee, unselected)`

Comment: *Level 1:* ————————————————————————————

Element: *The pick_up_schema's goal:*

`schema_goal(pick_up / coffee, pick_up)`

Element: *Nothing*

`schema_subgoals(pick_up / coffee, [])`

Element: *The pick_up_schema's initial selection status:*

`schema_status(pick_up / coffee, unselected)`

Element: *The put_down_schema's goal:*

`schema_goal(put_down / coffee, put_down)`

Element: *Nothing*

`schema_subgoals(put_down / coffee, [])`

Element: *The put_down_schema's initial selection status:*

`schema_status(put_down / coffee, unselected)`

Element: *The pour_schema's goal:*

`schema_goal(pour / coffee, pour)`

Element: *Nothing*

`schema_subgoals(pour / coffee, [])`

Element: *The pour_schema's initial selection status:*

`schema_status(pour / coffee, unselected)`

Element: *The open_schema's goal:*

`schema_goal(open / coffee, open)`

Element: *Nothing*

`schema_subgoals(open / coffee, [])`

Element: *The open_schema's initial selection status:*

`schema_status(open / coffee, unselected)`

Comment: *for phone answering*

Comment: *Level 2:*————————————————————

Element: *answer_phone's goal*

`schema_goal(answer_phone / phone, answer_phone)`

Element: *anser_phone's subgoals*

`schema_subgoals(answer_phone / phone, [pick_up_receiver, talking, hang_up])`

Element: *The answer_phone's order constraint:*

ordering_constraint(answer_phone / phone, pick_up_receiver < talking)

Element: *The answer_phone's order constraint:*

ordering_constraint(answer_phone / phone, talking < hang_up)

Element: *The answer_phone's initial selection status:*

schema_status(answer_phone / phone, unselected)

Comment: *Level 1:*————————————————————————

Element: *Nothing*

schema_goal(pick_up / phone, pick_up_receiver)

Element: *Nothing*

schema_subgoals(pick_up / phone, [])

Element: *Nothing*

schema_status(pick_up / phone, unselected)

Element: *Nothing*

schema_goal(hold_on / phone, talking)

Element: *Nothing*

schema_subgoals(hold_on / phone, [])

Element: *Nothing*

schema_status(hold_on / phone, unselected)

Element: *Nothing*

schema_goal(hang_up / phone, hang_up)

Element: *Nothing*

schema_subgoals(hang_up / phone, [])

Element: *Nothing*

schema_status(hang_up / phone, unselected)

Current Contents (of Schema States):

0 : schema_status(hang_up / phone, unselected)
0 : schema_subgoals(hang_up / phone, [])
0 : schema_goal(hang_up / phone, hang_up)
0 : schema_status(hold_on / phone, unselected)
0 : schema_subgoals(hold_on / phone, [])
0 : schema_goal(hold_on / phone, talking)
0 : schema_status(pick_up / phone, unselected)
0 : schema_subgoals(pick_up / phone, [])
0 : schema_goal(pick_up / phone, pick_up_receiver)
0 : schema_status(answer_phone / phone, unselected)
0 : ordering_constraint(answer_phone / phone, talking < hang_up)
0 : ordering_constraint(answer_phone / phone, pick_up_receiver < talking)
0 : schema_subgoals(answer_phone / phone, [pick_up_receiver, talking, hang_up])
0 : schema_goal(answer_phone / phone, answer_phone)
0 : schema_status(open / coffee, unselected)
0 : schema_subgoals(open / coffee, [])
0 : schema_goal(open / coffee, open)
0 : schema_status(pour / coffee, unselected)
0 : schema_subgoals(pour / coffee, [])
0 : schema_goal(pour / coffee, pour)
0 : schema_status(put_down / coffee, unselected)
0 : schema_subgoals(put_down / coffee, [])
0 : schema_goal(put_down / coffee, put_down)
0 : schema_status(pick_up / coffee, unselected)
0 : schema_subgoals(pick_up / coffee, [])
0 : schema_goal(pick_up / coffee, pick_up)
0 : schema_status(milk_from_carton / coffee, unselected)
0 : ordering_constraint(milk_from_carton / coffee, pour < put_down)

0 : ordering_constraint(milk_from_carton / coffee, open < pour)

0 : schema_subgoals(milk_from_carton / coffee, [pick_up, open, pour, put_down])

0 : schema_goal(milk_from_carton / coffee, add_milk)

0 : schema_subgoals(coffee_from_jar / coffee, [])

0 : schema_status(coffee_from_jar / coffee, unselected)

0 : schema_goal(coffee_from_jar / coffee, add_coffee)

0 : schema_status(coffee_from_pack / coffee, unselected)

0 : schema_subgoals(coffee_from_pack / coffee, [])

0 : schema_goal(coffee_from_pack / coffee, add_coffee)

0 : schema_status(sugar_from_bowl / coffee, unselected)

0 : schema_subgoals(sugar_from_bowl / coffee, [])

0 : schema_goal(sugar_from_bowl / coffee, add_sugar)

0 : schema_status(sugar_from_pack / coffee, unselected)

0 : schema_subgoals(sugar_from_pack / coffee, [])

0 : schema_goal(sugar_from_pack / coffee, add_sugar)

0 : schema_status(prepare_coffee / coffee, unselected)

0 : ordering_constraint(prepare_coffee / coffee, add_sugar < add_milk)

0 : ordering_constraint(prepare_coffee / coffee, add_coffee < add_milk)

0 : schema_subgoals(prepare_coffee / coffee, [add_sugar, add_coffee, add_milk])

0 : schema_goal(prepare_coffee / coffee, prepare_coffee)

Box Name: Trigger Schemas

Updates schemas activation values at each cycle.

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted): *Rule1 excites schema triggered by SAS*

IF: `task(prepare_coffee, 10)` is in *Experimental World:World State*
`schema(prepare_coffee)` is in *Temporary Schema*

THEN: send `excite(prepare_coffee / coffee, 1)` to *Schema Hierarchy*

Rule 2 (unrefracted): *Rule1 excites schema triggered by SAS*

IF: `task(answer_phone, 30)` is in *Experimental World:World State*
`schema(answer_phone)` is in *Temporary Schema*

THEN: send `excite(answer_phone / phone, 1)` to *Schema Hierarchy*

Rule 3 (unrefracted): *Activation flow from parent*

IF: `selected_parent_of(S, P)`
`ordering_constraints_satisfied(S, P)`
`node(P, V)` is in *Schema Hierarchy*
`number_of_children(P, N)`
`Value is (V) / (N)`

THEN: send `excite(S, Value)` to *Schema Hierarchy*

Rule 4 (unrefracted): *Inhibit a schema after its goal is achieved*

IF: `node(Schema, Value)` is in *Schema Hierarchy*
`goal_of_schema_is_achieved(Schema)`

THEN: send `excite(Schema, -1)` to *Schema Hierarchy*

Rule 5 (unrefracted): *Inhibit the former schema during interruptions*

IF: `task(T, U)` is in *Experimental World:World State*
`schema_goal(Name, T)` is in *Schema States*
`not schema(T)` is in *Temporary Schema*

THEN: send `excite(Name, -1)` to *Schema Hierarchy*

Rule 6 (refracted): *Continue former task from break action after interruption*

IF: `finish_action(Goal)` is in *Experimental World:World State*

`schema_subgoals(prepare_coffee / coffee, Subgoals)` is in *Schema States*

`Goal` is in *a member of Subgoals*

`schema_goal(Schema, Goal)` is in *Schema States*

THEN: send `goal_of_schema_is_achieved(Schema)` to *Trigger Schemas*

Condition Definition: *goal_of_schema_is_achieved/1: Has this schema's goal been achieved?*

goal_of_schema_is_achieved(Schema):-

`schema_status(Schema, selected, Subgoals, Time)` is in *Schema States*

not - *Goal* is a `member of Subgoals`

goal_of_schema_is_achieved(Schema):-

`schema_status(Schema, unselected)` is in *Schema States*

`schema_status(Parent, selected, Subgoals, Time)` is in *Schema States*

schema_goal(Schema, Goal) is in *Schema States*

\+ *Goal* is a `member of Subgoals`

not - *Goal* is a `member of Subgoals`

Condition Definition: *ordering_constraints_satisfied/2: Are a schema's preconditions met?*

ordering_constraints_satisfied(Schema, Parent):-

`schema_goal(Schema, Goal)` is in *Schema States*

`schema_status(Parent, selected, SubGoals, Time)` is in *Schema States*

not ordering_constraint(Parent, PreGoal ¡ Goal) is in *Schema States*

- *PreGoal* is a `member of Subgoals`

Condition Definition: *number_of_children/2: How many subgoals does a schema have?*

number_of_children(Schema, N):-

`schema_subgoals(Schema, SubGoals)` is in *Schema States*

N is `the length of Subgoals`

Condition Definition: *schema_children/2: Get the list of child goals of a schema.*

schema_children(Schema, Children):-

`schema_subgoals(Schema, ChildGoalList)` is in *Schema States*

strip_functions(ChildGoalList, Children)

schema_children(Schema, []):-

node(Schema, X) is in *Schema Hierarchy*

not schema_subgoals(Schema, ChildGoalList) is in `Schema States`

Condition Definition: *Strip off any argument function specifications from a goal list.*

strip_functions([], []).

strip_functions([Goal / _—Rest], [Goal—StripRest]):-

!

strip_functions(Rest, StripRest)

strip_functions([Goal—Rest], [Goal—StripRest]):-

strip_functions(Rest, StripRest)

Condition Definition: *selected_parent_of(Schema, Parent):-*

strip_functions([Goal—Rest], [Goal—StripRest]):-

parent_of(Schema, Parent)

schema_status(Parent, selected, _, _) is in `Schema States`

Condition Definition: *parent_of/2: Check schema/parent relationships*

parent_of(Schema, Parent):-

schema_children(Parent, ChildGoals)

schema_goal(Schema, Goal) is in `Schema States`

Goal is `a member of ChildGoals`

Condition Definition: *parent_of/2: Check schema/parent relationships*

parent_of(Schema, Parent):-

schema_children(Parent, ChildGoals)

schema_goal(Schema, Goal) is in `Schema States`

Goal is `a member of ChildGoals`

Box Name: Select & Act

Handle schema selection and action execution.

Class: Box/Process

Rules & Condition Definitions:

Rule 1 (unrefracted): *Select any appropriate schemas*

IF: `schema_goal(Name, Goal)` is in *Schema States*

`schema_should_be_selected(Name, UnachievedSubgoals)`

`the current cycle` is *Time*

THEN: delete `schema_status(Name, unselected)` from *Schema States*

add `schema_status(Name, selected, UnachievedSubgoals, Time)` to *Schema States*

add *is_doing(Goal)* to *Experimental World:World State*

Rule 2 (refracted): *Deselect any appropriate schemas*

IF: `schema_status(Name, selected, Subgoals, Time)` is in *Schema States*
`schema_should_be_deselected(Name)`

THEN: delete `schema_status(Name, selected, Subgoals, Time)` from *Schema States*
add `schema_status(Name, unselected)` to *Schema States*

Rule 3 (unrefracted): *Update achieved subgoals*

TRIGGER: `done(Goal)`

IF: `schema_status(Name, selected, BeforeSubgoals, Time)` is in *Schema States*
`mark_goal(Goal, BeforeSubgoals, AfterSubgoals)`
not `not schema_should_be_deselected(Name)`

THEN: delete `schema_status(Name, selected, BeforeSubgoals, Time)` from *Schema Sta*
add *schema_status(Name, selected, AfterSubgoals, Time)* to *Schema States*

Rule 4 (unrefracted): *Recurse on the parent if all subgoals are achieved*

TRIGGER: `done(Goal)`

IF: `schema_status(Name, selected, BeforeSubgoals, Time)` is in *Schema States*
`mark_goal(Goal, BeforeSubgoals, AfterSubgoals)`
not - `OtherGoal is a member of AfterSubgoals`
`schema_goal(Name, ParentGoal)` is in *Schema States*

THEN: send `done(ParentGoal)` to *Select & Act*
add `finish_action(ParentGoal)` to *Experimental World:World State*

Rule 5 (refracted): *Execution an action when a lowest-level schema is selected*

IF: `schema_status(S, selected, [], Time)` is in *Schema States*

`schema_goal(S, G)` is in *Schema States*

THEN: add `finish_action(G)` to *Experimental World:World State*

send `done(G)` to *Select & Act*

Condition Definition: *mark_goal/3: Take a list of goals and mark one as achieved*

mark_goal(Goal, [- Goal—Rest], [+ Goal—Rest]):-

!

mark_goal(Goal, [H—T], [H—MT]):-

`mark_goal(Goal, T, MT)`

Comment: `Conditions involved in schema selection/deselection:`

Condition Definition: *schema_should_be_selected/2: True for unselected winning schemas with activation above threshold*

schema_should_be_selected(Schema, Subgoals):-

`schema_status(Schema, unselected)` is in *Schema States*

`activation_above_threshold(Schema)`

winning_schema(Schema)

schema_children(Schema, Children)

construct_unachieved_list(Children, Subgoals)

Condition Definition: *construct_unachieved_list/2: Initially, all subgoals are unachieved.*

number_of_children(Schema, N):-

`schema_subgoals(Schema, SubGoals)` is in *Schema States*

N is `the length of Subgoals`

Condition Definition: *schema_children/2: Get the list of child goals of a schema.*

construct_unachieved_list([], []).

construct_unachieved_list([G—T0], [- G—T1]):-

construct_unachieved_list(T0, T1)

Condition Definition: *schema_should_be_deselected/1: True for selected schemas that are no longer winning*

schema_should_be_deselected(Schema):-

`schema_status(Schema, selected, _, _)` is in *Schema States*

not *not winning_schema(Schema)*

schema_should_be_deselected(Schema):-

schema_status(Schema, selected, _, _) is in *Schema States*

winning_schema(Schema)

`selected_parent_of(Schema, ParentSchema)`

`schema_should_be_deselected(ParentSchema)`

schema_should_be_deselected(Schema):-

schema_status(Schema, selected, _, _) is in *Schema States*

winning_schema(Schema)

not `selected_parent_of(Schema, _)`

`parent_of(Schema, ParentSchema)`

`schema_should_be_selected(ParentSchema, _)`

Condition Definition: *winning_schema/1: A schema is the winner if no competitor has greater activation:*

winning_schema(Schema1):-

`node(Schema1, Activation1)` is in *Schema Hierarchy*

not *competitor_of_schema(Schema1, Schema2)*

`node(Schema2, Activation2)` is in *Schema Hierarchy*

`Activation2` is greater than *Activation1*

!

Condition Definition: *activation_above_threshold/1: Is a schema's activation above the selection threshold?*

activation_above_threshold(Schema):-

node(Schema, Activation) is in *Schema Hierarchy*

Activation is not less than `0.60`

Condition Definition: *competitor_of_schema/2: Schemas compete if they achieve the same goal, share a subgoal, or share resource requirements*

competitor_of_schema(Schema, Competitor):-

schemas_share_goal(Schema, Competitor)

Schema is distinct from `Competitor`

competitor_of_schema(Schema, Competitor):-

schemas_share_subgoal(Schema, Competitor)

not *schemas_share_goal(Schema, Competitor)*

Condition Definition: *schemas_share_goal/2: Do two schemas achieve the same goal?*

schemas_share_goal(Schema1, Schema2):-

schema_goal(Schema1, Goal) is in *Schema States*

schema_goal(Schema2, Goal) is in `Schema States`

Condition Definition: *schemas_share_subgoal/2: Schemas share a subgoal is their subgoal lists have a common element*

schemas_share_subgoal(Schema1, Schema2):-

schema_children(Schema1, Children1)

schema_children(Schema2, Children2)

common_member(Children1, Children2)

Condition Definition: *common_member/2: Do two lists have a common member?*

common_member([Goal—_], L):-

`Goal` is *a member of L*

!

common_member([_—Rest], L):-

`common_member(Rest, L)`

Condition Definition: *schema_children/2: Get the list of child goals of a schema.*

schema_children(Schema, Children):-

`schema_subgoals(Schema, ChildGoalList)` is in *Schema States*

strip_functions(ChildGoalList, Children)

schema_children(Schema, []):-

`node(Schema, X)` is in *Schema Hierarchy*

not `schema_subgoals(Schema, ChildGoalList)` is in *Schema States*

Condition Definition: *Strip off any argument function specifications from a goal list.*

strip_functions([], []).

strip_functions([Goal / _—Rest], [Goal—StripRest]):-

!

strip_functions(Rest, StripRest)

strip_functions([Goal—Rest], [Goal—StripRest]):-

strip_functions(Rest, StripRest)

Condition Definition: *selected_parent_of/2: Which (if any) superschema of the given schema is selected?*

selected_parent_of(Schema, Parent):-

parent_of(Schema, Parent)

`schema_status(Parent, selected, _)` is in *Schema States*

Condition Definition: *parent_of/2: Check schema/parent relationships*

parent_of(Schema, Parent):-

schema_children(Parent, ChildGoals)

`Goal` is *a member of ChildGoals*

Bibliography

[1] D. Rice, P. J. Fox, and W. Max, "The economic burden of alzheimer's disease care," *Health Affairs*, vol. 12:2, pp. 164–176, Summer 1993.

[2] D. A. Norman and T. Shallice, "Attention to action: Willed and automatic control of behavior," in *Consciousness and Self-Regulation* (R. J. Davidson, G. E. Schwartz, and D. Shapiro, eds.), New York: Plenum Press, 1980.

[3] R. Cooper and J. Fox, "COGENT: A visual design environment for cognitive modelling," *Behavior Research Methods, Instruments, and Computers*, vol. 30, no. 4, pp. 553–564, 1998.

[4] D. Shenk, *The Forgetting: Alzherimer's: Portrait of an Epidemic.* New York: Doubleday, September 2001.

[5] V. Rialle, N. N. F. Duchêne, L. Bajolle, and J. Demongeot, "Health 'smart' home: Information technology for patients at home," *Telemedicine Journal and E-Health*, vol. 8, no. 4, pp. 395–409.

[6] G. Elger and B. Furugren, "SmartBo – An ICT and computer-based demonstration home for disabled," in *Proc. 3rd TIDE Congress: Technology for Inclusive Design and Equality Improving the Quality of Life for the European Citizen*, (Marina Congress Center, Helsinki, Finland), June 23-25 1998.

[7] G. Elger, "'SmartBo' – A smart house for people with disabilities," tech. rep., The Swedish Handicap Institute, P.O. Box 510, SE-162 15 Välingby, Sweden, 2002.

[8] H. Pigot, B. Lefebvre, J.-G. Meunier, B. Kerhervé, A. Mayers, and S. Giroux, "The role of intelligent habitats in upholding elders in residence," in *5th international conference on Simulations in Biomedicine*, (Slovenia), pp. 497–506, April 2-4 2003.

[9] H. Pigot, A. Mayers, and S. Giroux, "The intelligent habitat and everyday life activity support," in *5th International Conference on Simulations in Biomedicine*, (Slovenia), pp. 507–516, April 2-4 2003.

[10] H. Kautz, D. Fox, O. Etzioni, G. Borriello, and L. Arn, "An overview of the assisted cognition project," in *AAAI-2002 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, (Edmonton, Alberta, Canada), 29 July 2002.

[11] S. Grant, "Developing cognitive architecture for modeling and simulation of cognition and error in complex tasks," in *RoHMI project meeting*, (Valenciennes, France), February 1996.

[12] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR: an architecture for general intelligence," *Artificial Intelligence*, vol. 33, pp. 1–64, September 1987.

[13] A. Newell, *Unified Theories of Cognition.* Cambridge, MA: Harvard University Press, 1990.

[14] J. R. Anderson and C. Lebiere, *The Atomic Components of Thought.* Mahwah, New Jersey: Lawrence Erlbaum Associates, June 1998.

[15] J. R. Anderson, *The Architecture of Cognition.* Cambridge MA: Harvard University Press, March 1986.

[16] W. Schneider and W. Oliver, "An instructable connectionist/control architecture: Using rule-based instructions to accomplish connectionist learning in a human time scale," in *Architectures for intelligence: The 22nd Carnegie Mellon symposium on cognition* (K. V. Lehn, ed.), (Hillsdale, NJ: Erlbaum), pp. 113–145, 1991.

[17] D. E. Meyer and D. E. Kieras, "A computational theory of executive cognitive processes and multiple-task performance: I. Basic mechanisms," *Psychological Review*, vol. 104, no. 1, pp. 3–65, 1997.

[18] H. D. G. and B. M. D., "An act-r/pm model of the articulatory loop," in *5th International Conference on Cognitive Modeling*, (Bamberg, Germany), pp. 135–140, April 10-12 2003.

[19] R. Cooper and T. Shallice, "Contention scheduling and the control of routine activities," *Cognitive Neuropsychology*, vol. 17, no. 4, pp. 297–338, 2000.

[20] T. Shallice and P. Burgess, "The domain of supervisory processes and temporal organization of behaviour," *Philosophical Transactions of the Royal Society of London B*, vol. 351, pp. 1405–1412, 1996.

[21] B. Ivan, *Prolog Programming for Artificial Intelligence*. Wokingham, UK: Addison-Wesley, 2000.

[22] P. Yule and R. Cooper, "The cogent tutorial," in *22 Annual Conference of the Cognitive Science Society*, (Philadelphia, USA), August 2000.

[23] D. W. Glasspool, "The integration and control of behaviour: Insights from neuroscience and AI," in *How to design a functional mind*, AISB Convention, April 2000.

[24] S. K. Das, J. Fox, D. Elsdon, and P. Hammond, "A flexible architecture for autonomous agents," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 407–440, October 1997.

[25] R. P. Cooper, P. Yule, and J. Fox, *Modelling High-Level Cognitive Processes*. Mahwah, New Jersey: Lawrence Erlbaum Associates, 2002.

[26] Y. Zhang, H. Pigot, and A. Mayers, "Attention switching during interruptions," in *International Conference on Machine Learning and Cybernetics 2004*, (Shanghai, China), August 26-29 2004.

[27] M.-H. Sohn and J. R. Anderson, "Task preparation and task repetition: Two-component model of task switching," *Journal of Experimental Psychology: General*, vol. 130, pp. 764–778, 2001.

[28] J. R. Anderson, M. Matessa, and C. Lebiere, "Act-r: A theory of higher level cognition and its relation to visual attention," *Journal of Human-Computer Interaction*, vol. 12, pp. 439–462, 1997.

[29] J.Bryson and L. A. Stein, "Modularity and design in reactive intelligence," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, (Seattle, Washington, USA), pp. 1115–1120, 2001.

[30] E. Gat, *Artificial intelligence and mobile robots: case studies of successful robot systems*, pp. 195 – 210. Cambridge, MA, USA: MIT Press, 1998.

[31] E. M. Altmann and J. G. Trafton, "Memory for goals: an activation-based model," *Cognitive Science*, vol. 26, pp. 39–83, 2002.

[32] G. L. Steele, *Common Lisp: The Language*. Woburn, USA: Digital Press, 1990.

[33] R. Cooper, T. Shallice, and J. Farringdon, *Symbolic and continuous processes in the automatic selection of actions in Hybrid Problems, Hybrid Solutions*, pp. 22–37. Amsterdam: IOS Press, 1995.