

Modelling Psycho-physiological effects in  
Computer Generated Forces  
Master Thesis <sup>1</sup>

Niklas Wallin

25th June 2002

<sup>1</sup>This thesis corresponds to the effort of 20 full-time working weeks

## **Abstract**

This is a report from a master thesis project, presenting methods of creating Computer Generated Forces, CGF's. The report is a documentation of an implementation of CGF's and describes methods that may be useful when creating synthetic entities. Methods of structuring tasks, environments, rational and human-like behaviour are presented. Tools for creating CGF or automated behaviour are described as well as a comparison between two such architectures.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Purpose . . . . .	1
1.3	Methodology . . . . .	2
1.4	Outline . . . . .	3
<b>2</b>	<b>Scenario</b>	<b>4</b>
2.1	Agents . . . . .	4
2.2	CGF . . . . .	5
2.3	Scenario . . . . .	5
2.4	Environment . . . . .	7
2.5	Task analysis . . . . .	8
<b>3</b>	<b>Modelling human behaviour</b>	<b>13</b>
3.1	Modelling . . . . .	13
3.2	Cognitive architectures . . . . .	14
3.2.1	Introduction . . . . .	14
3.2.2	Soar . . . . .	17
3.2.3	Agent Factory . . . . .	19
3.2.4	Comparison . . . . .	25
3.3	Human psychology . . . . .	28
3.3.1	Psychological models . . . . .	28
3.3.2	Attention . . . . .	33
3.3.3	Visual attention . . . . .	34
3.3.4	Shifting focus of attention . . . . .	36
3.3.5	Attention and psycho-physiological indices . . . . .	39
3.3.6	Situational awareness . . . . .	39
<b>4</b>	<b>Implementation</b>	<b>41</b>
4.1	Soar agent . . . . .	41
4.1.1	Platforms . . . . .	41
4.1.2	Agent . . . . .	44
4.1.3	Future enhancements . . . . .	48

4.2	Agent Factory agent . . . . .	49
4.2.1	Platforms . . . . .	49
4.2.2	Agent . . . . .	51
<b>5</b>	<b>Conclusions</b>	<b>52</b>
<b>A</b>	<b>Soar agent</b>	<b>55</b>
A.1	Goal hierarchy . . . . .	55
A.2	Input-link . . . . .	56
A.3	System . . . . .	56

# Chapter 1

## Introduction

### 1.1 Background

Computer based simulations, here referred to as simulations, provide great opportunities in favour of traditional methods of conducting live exercises and man in the loop simulations. Live military exercises and training activities that once were not possible or needed months of preparation, can now be simulated within hours at much lower costs.

Evaluations of alternative tactics, doctrine, environments or system components are other activities that modelling and simulation are often dedicated to. To be able to take advantage of the features of simulation, the simulated activities must of course be supplements to live exercises. It is therefore necessary to develop realistic simulations that realise those demands. For military training purposes, simulated environments often consist of many entities that participate on several political sides. To avoid problems such as lack of resources, preparation and cost when conducting large exercises, it is desirable to automate some or all of the participating entities. Further, the need for realistic simulations require realistic automated behaviour. By implementing such automated behaviour, the capacity of the simulation system increases and scenarios that are more complex are achievable. It is also possible for live simulation and computer based simulations to work together in exercises.

Aids for decision-making and information processing may also require automated behaviour during real operations.

### 1.2 Purpose

This master thesis project will examine some aspects of implementing computer generated forces (CGF's) and human behaviour representation (HBR) for use in simulations. The work presented is part of a larger project per-

formed at the Swedish Defence Research Agency, FOI<sup>1</sup>. A continuing goal of the project is to model human behaviour representations of entities in different domains and at different levels of aggregation. More contiguous goals are to evaluate different tools, define a method for documentation, a method for development and to create representations that will act in air combat simulations.

The purpose of the thesis is to serve as pre-study. The study will address some of the goals presented, to give an insight in CGF modelling; techniques, problems, tools, representations, etc.

### 1.3 Methodology

A CGF is an automated entity or agent that acts in simulated combat situations. The requirements on the CGF's in this study are that they should show human-like behaviour. Therefore the study has been involved with examining theories of cognition and explanation of cognitive activities. The mainstream of cognitive theories has long been rooted in a rationalistic tradition, but have lately been challenged by other, although not new, sources of understanding [win86]. The results and discussions in this study build upon the older, rationalistic view to produce behaviour in an ideal case. The choice to follow the rationalistic stream is mainly due to its influence on western science and technology in the past. Since all well-known and practised methods of modelling, validation, verification and task analysis follow the old tradition, it was a natural choice. However, rational behaviour is not regarded as human-like behaviour in this case. Rational behaviour is something that a human or any other organism is thought or strive to perform. One way to express rational behaviour would then be to exactly follow a manual or taught instructions. Human behaviour is produced when rational behaviour must stand back for cognitive limitations inherited by humans. The rational behaviour is supposed to be presented for experts in the corresponding domain to validate and assess. By adding human-like constraints on performance the CGF's will produce more human-like behaviour. By inheriting both behaviour models, the CGF's will act in a more realistic manner. The level of detail depends on the role of the CGF and the purpose of the simulation. Some assumptions and simplifications are introduced to clarify the computational constraints compared to its physical counterpart.

From the rational behaviour view, the purpose is to explore techniques for acquiring knowledge from subject matter experts (SME's) in the current domain and to translate that knowledge into rational behaviour inherited by the CGF's.

When looking at human behaviour representation, one of the main purposes is to explore the possibilities of adding representations founded on data

---

<sup>1</sup><http://www.foi.se>

attained from experiments with human participants. Complex relationship models of psycho-physiological and psychological factors are available from other studies and theories. By using these psychological models, would it be feasible to create CGF's with more realistic behaviour?

There are several tools or architectures designed to aid in the work of creating behaviour representations. In this project, two architectures will be studied and used for implementation of CGF's. From a modeller's perspective, a comparison will be made to highlight some of the desirable and non-desirable features of the two architectures.

The study will investigate some of the techniques that are used and propose some ideas and considerations on modelling human behaviour. For a better understanding of the aspects that appear when modelling CGF's, an implementation will be created in parallel with the discussions. The intention of this paper is not to propose a strategy or guideline for creating CGF's; it is simply a description of the development of one such CGF.

## 1.4 Outline

Chapter 2 describes the construction of a scenario or context where the CGF will act. This chapter also include task analyses and some aspects of gathering data, with the aid of experts, to represent rational behaviour.

Chapter 3 discusses implementation techniques. Two cognitive architectures are studied and some techniques to extend those architectures are proposed. The proposed extensions are supposed to create behaviour that is more realistic by adding psychological models from studies on human counterparts.

Chapter 4 describes an implementation of a CGF.

Chapter 5 reviews some of the results and recommendations for future developments.

# Chapter 2

## Scenario

### 2.1 Agents

*“An entity that resides in environments where it interprets sensor data that reflect events in the environment and executes “motor” commands that produce effects in the environment.”*

FIPA Rationale 1996  
Foundation for Intelligent Physical Agents

FIPA extends the above definition of an agent by adding these optional attributes:

**Autonomy.** Agents can operate without the direct intervention of humans or others.

**Social ability.** Agents can interact with other agents and/or humans.

**Reactivity.** Agents perceive their environment and respond in a timely fashion.

**Pro-activeness.** Agents can exhibit goal-directed behaviour by taking initiatives.

**Mobility.** Agents can move to other environments.

**Temporal continuity.** Agents are continuously running processes.

**Adaptivity.** Agents automatically adapt to changes in their environment.

Nowadays, the term agent is widely used and the definitions may vary. A general, but not concise, definition is a software component that can act autonomously and pro-actively. Agents are often used as aiding tools in a wide variety of applications. Mobile agents are typically used to gather

information from different distributed sources and compile that information in an appropriate way. Other roles for agents are as stand-in opponents or team players for training and analysis purposes. For most applications, the agents should show rational behaviour and as stand-ins, they should show human-like behaviour. The agents, developed in this study are autonomous, reactive, pro-active and adaptive. The agents reside in an environment where they can perceive objects and respond to changes in that environment. Once started, they can operate without further guidance. Whether an agent is social or not depends on the definition. Here, the agents will react on how other agents make changes to the environment. No direct communication between agents will be implemented. As the agents will act as stand-ins, they will also show a certain degree of rational human-like behaviour.

## 2.2 CGF

*“A generic term used to refer to computer representation of entities in simulations which attempts to model human behaviour sufficiently so that forces will take some actions automatically.”*

Modelling and Simulation Master Plan  
US Department of Defence Oct 1995

The CGF's and their behaviour are not particularly useful themselves [Tol]. They are only meaningful within a context as a component of a larger system or simulation. There is a need for a synthetic environment where the CGF's can act, as its corresponding physical entity. In that environment, the CGF's should be able to post actions as well as gather information. The environment itself can consist of several components with descriptions of other entities, terrain, weather, etc.

The above definition of Computer Generated Forces [Doma] is not necessarily restricted to military entities, but also civilian engagement in crisis and conflict situations. However, the need for intelligent human behaviour does not only apply to military and civilian conflict situations. Therefore, the more general term agent is used in the remainder of this report even though the implementation corresponds to a CGF.

## 2.3 Scenario

To make a better understanding of modelling agents, a model was created in a systematic manner while looking at different modelling techniques. As a first step, one must classify the role of the agent [JERTb]. What should the agent be able to do? Where should it do it? How should it do it?

Most military operations are described as missions. Each mission is equipped with resources that are needed to complete it, a detailed plan of

how the mission should be performed and a goal or purpose. For instance, a typical air mission is to escort a tanker to protect it from hostile engagement. The mission goal is to keep the tanker alive. The escort group (resources) consists of four fighters (military aircrafts equipped and designed for air-to-air engagements). The plan holds descriptions of the flight path, formation, time, rules of engagement, etc. The pilots create the plan before the mission, often by mental flying to try to cover all situations that may appear.

The pilot model presented here was acting as a military pilot conducting a strike mission. Strike missions are quite straightforward to describe, although not so easy to perform. To add more interaction a second pilot agent, conducting a patrol mission, was created. Both agents acted in simulated cockpit environments where they read displays and managed their corresponding joysticks. The cockpit was part of an aircraft that also acted in an environment consisting of a terrain map, ground radars, and other aircrafts. All information that was available to the agents was conveyed by their cockpits and all interactions were done solely through the joystick.

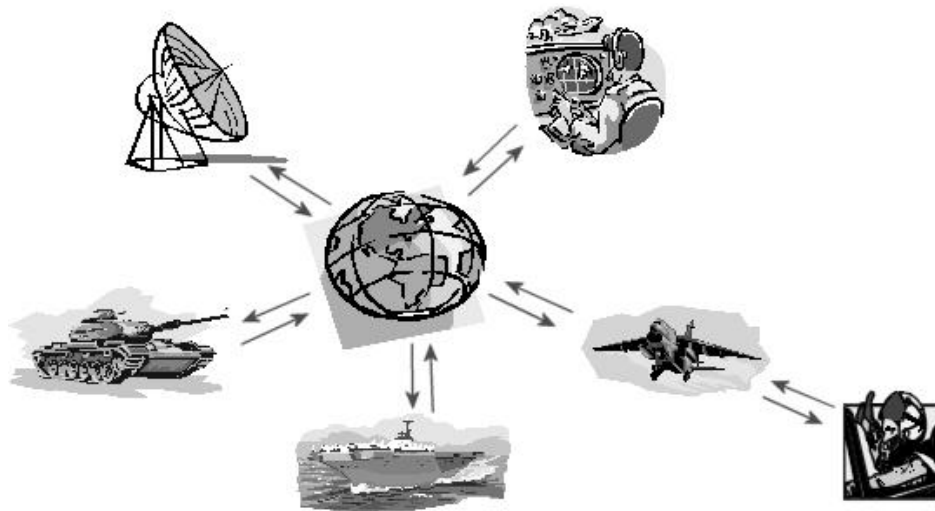


Figure 2.1: A typical simulation environment. The platforms run their own simulation systems and can interact with each other. The agent interacts only through the interface provided by its platform.

The strike mission was outlined as follows: Navigate through five way-points, in order one to five, on the map. Way-point number five is located at the target's position. When way-point number four is reached, initiate the attack. As the target is approached, make necessary manoeuvres to achieve optimal firing position. When the target has been reached fly to way-point number one. If any hostile threats are seen on radar, ignore them and con-

tinue to execute the mission.

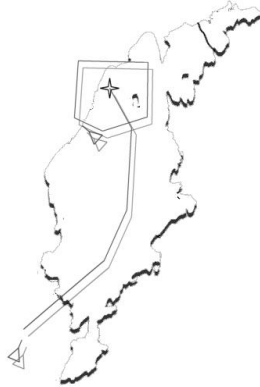


Figure 2.2: Scenario map. A strike mission and a patrol mission.

The patrol mission was outlined as follows: Navigate through five way-points in order one to five. When way-point number five is reached fly to way-point number one and repeat the cycle. Continue to fly this pattern until a hostile aircraft appears on radar. If a hostile aircraft appears within a predefined radius, start to pursue that aircraft.

## 2.4 Environment

A cockpit of a military aircraft is a very complex environment with a vast amount of information. An example is the American fighter F/A-18 Hornet with over a hundred different indicators and about 60 different display modes containing 177 symbols and 675 acronyms. Of course it is impossible for even the most experienced pilot to keep track of all symbols at all times. Studies have shown that pilot performance decrease rapidly when the amount of information increases [SAT<sup>+</sup>92] [SF99] [Ber00]. To avoid reduced performance much effort has been made to develop better methods to convey this information and to allow the pilot to focus on activities that are more important. The interface between the aircraft and the pilot consists of display monitors, visible and audible warning indicators, buttons, joystick, throttle, rudder pedals, etc. To build a realistic model of a cockpit, all of these components should be included in the interface. The model should however be adapted to the cockpit environment and the purpose of the simulation. In most cases, many components will have little effect on the internal behaviour of the pilot and even less effect on the other participants in the simulation. It is therefore appropriate to include the essential components that the agent should be able to monitor and affect during the simulation. This listing is

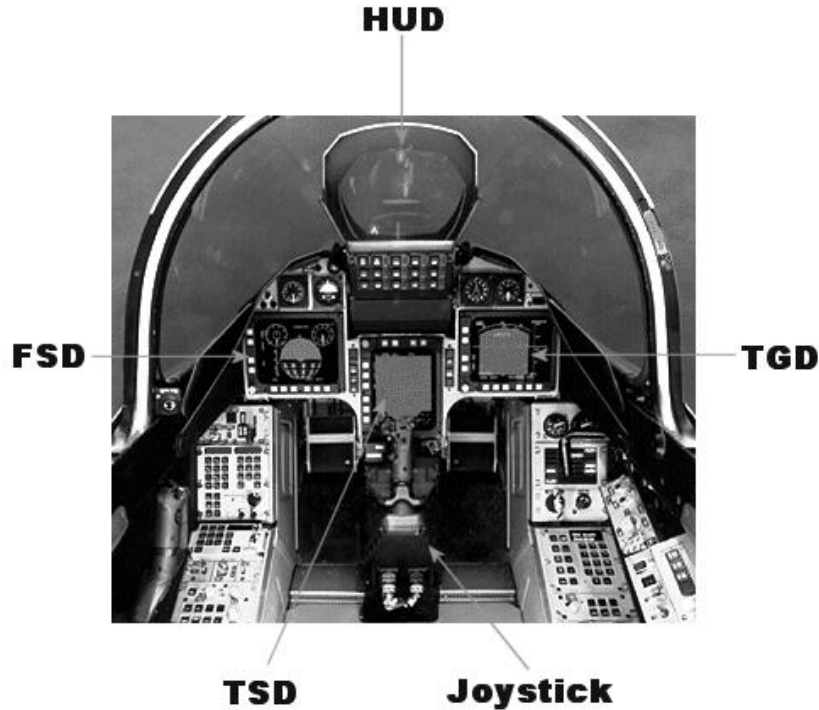


Figure 2.3: JAS-39 Cockpit. The four most important displays as well as the joystick are marked as the Heads Up Display (HUD), the Flight Situation Display (FSD), the Tactical Situation Display (TSD) and the Target Geometry Display (TGD).

then a description of the interface between the agent and the cockpit. Based on such documentation other aircraft models or other agent models could easily be made compatible with existing models. Although different types of aircrafts use different symbols and have different components most of them share some common components. Table 2.1 lists the components that the agents were capable to reason about in this study.

## 2.5 Task analysis

To create a realistic behaviour of the pilot, the mission or task of the modelled pilot must be analysed [Nev97]. Professionals or experts in the current domain best provide this knowledge but there are various ways to obtain this information. Discussions, interviews and careful studies of live situations are some ways to acquire information. Another popular method is to have experts sit around a table and actually manoeuvre plastic aircraft models in different scenarios. The data collected in this study was obtained with the

Component	Description	Unit	Display
Speed	Current speed	m/s	HUD, FSD
Heading	Current heading	degrees	HUD, FSD
Altitude	Current altitude	m	HUD, FSD
AOA	Angle of attack	degrees	HUD
Heading bug	Marker indicating the difference between the current heading and desired heading		HUD
Target-heading	Relative heading to target	degrees	TSD
Target-type	Aircraft, Radar, etc		TSD

Joystick

Table 2.1: Objects that a pilot agent can interact with.

aid of pilots from the Swedish Air Force at F17 in Blekinge, Sweden. At F17, there are highly sophisticated recording devices that continuously log flight parameters during missions. The missions can then be replayed at a simulator for post mission analysis. By using this simulator it was possible to see what the actual pilot saw on the cockpit displays while he was executing the mission. It was also possible to figure out some of the actions that followed when parameters in those displays changed. For example, it was quite easy to understand how some manoeuvres depended on the range to the target. To make the analyses even more effective, pilots were present that could explain the situations and how they reasoned in different phases.

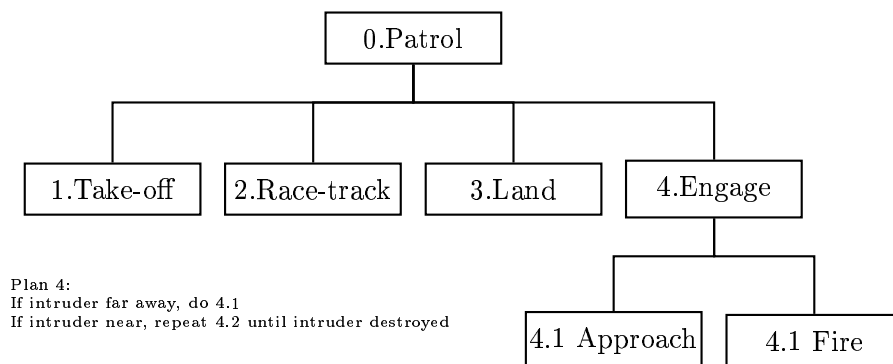


Figure 2.4: HTA for a patrol task. Operations are numbered and part of a plan to achieve a goal

Documentation was done based on a method called Hierarchical Task Analyses (HTA) [KA89]. HTA is a decomposition technique that is used to create descriptions of tasks in terms of goals, operations and plans (see figure 2.4). Goals are desired states of a system, for instance destruction of a target

or airspace free of hostile aircrafts. An operation is a unit of behaviour that is executed to attain a goal. This unit of behaviour can be described in different levels of complexity and duration. For example, in one context an operator could be to patrol an airfield, while another operator could be to move the joystick. Both operations could be part of a single goal but are described at different levels of detail. A plan states the condition that specifies when to perform each operation. Plans often include a sequence of actions related to time or process conditions. Sometimes high-level operations need to be broken down into sub-operations and thus be part of a sub-goal. The process of breaking down high-level operations into lower level operations continues until the required level of detail is attained. Many cognitive theories state that all organisms are continually trying to attain goals and that goals are part of higher goals. This is why HTA has been widely used in cognitive modelling and why it was chosen here.

The strike mission was divided into operations and sub operations and reviewed by the pilots. For every operation, a list of conditions and actions was attached to decompose the operation even further (see table 2.2). This analysis resulted in a detailed description of how and when the tasks were to be executed.

Figure 2.5 shows a part of the goal hierarchy, for the strike mission, that resulted from the HTA.

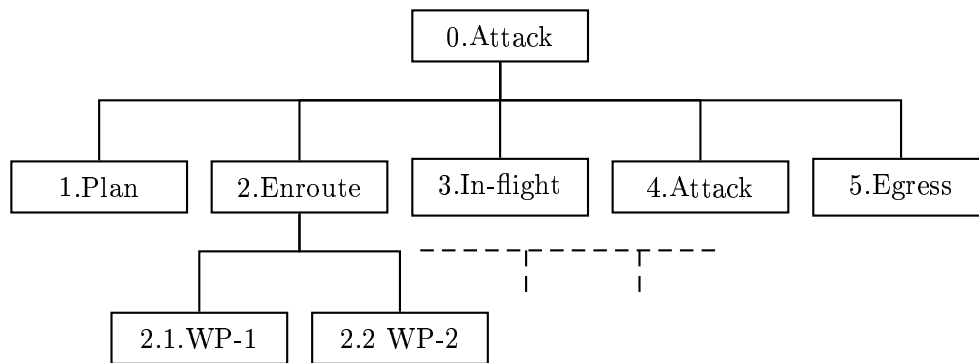


Figure 2.5: HTA for a strike task.

By using an efficient documentation method, that both experts and modellers can easily understand the collection of data and the validation processes improves [DS00]. Documentation also provides a layer between the experts and the implementation. This facilitates the creation of different models and implementations from a common set of data. The HTA that was used here provided a sufficient tool for documentation that experts and modellers could understand. It is also desirable to use a standard documentation technique that is applicable for entities in different domains. This gives an opportunity

<b>Description</b>	
Function	What is the purpose of the action?
<b>Requirements</b>	
Initiating cue/event	When should the task be initiated?
Skills	What cognitive skills are required?
<b>Hardware features</b>	
Controls used?	Which controls are used, how are they used?
Displays used?	Which displays are used?
Critical values	What are the critical values?
Aids	What aids are needed and possible during the task?
<b>Nature of the task</b>	
Actions required	What actions should be taken?
Decisions required	What decisions must be taken?
Responses required	To what and how should responses be executed?
Task criticality	How important/critical is the task?
<b>Performance</b>	
Time	What timing issues are there (min,avg,max)?
Accuracy	How accurate must the task be performed?
Speed	How fast must the task be performed?
<b>Other activities</b>	
Communication	Are there any grants/reports to/from other entities?
Concurrent tasks	Are there any concurrent tasks affected?
<b>Result</b>	
Output	What is the result/supposedly result?
Feedback	Are there any feedback for evaluation, etc?
<b>Consequences</b>	
Likely/typical errors	What can go wrong and how?

Table 2.2: Example of decomposition categories of an operation in HTA.

to create a library of data sets that can easily be maintained, updated and reviewed. HTA will certainly be a useful tool as a concept because it is easy to translate hierarchies to goal oriented cognitive architectures. By altering the decomposition categories, different domains and environments could be analysed.

## Chapter 3

# Modelling human behaviour

*“Human behaviour is purposive action of a human being to an idiosyncratic meaningful situation”*

Simulation of and for Military Decision Making  
RTO SAS Lecture Series

### 3.1 Modelling

In agent development, there are two main approaches. As mentioned in 1.3 the models should act like humans or they should act rationally. In this paper, rational behaviour is the behaviour that may be predicted and expected from an organism in that particular situation. Rational behaviour could be written as a manual or a book of rules that the organism should follow. Human behavior is constructed from rational behaviour but also from reflections of human physical and/or psychological constraints. The difference is that humans do not always practice the most appropriate reaction to a situation whereas a rational model is more of an expert system that lacks human constraints. For example, the most rational thing to do is to duck when an object is flying towards the head of a human. This could be written as a rule or a set of rules in a “how to survive” guide. Although most people know this rule and that it is the most appropriate thing to do, accidents still happens. Because of long reaction times or the fact that some people become paralyzed in dangerous situations, the actions does not always correspond to the most expected output or actions.

In this project, human behaviour models will be considered. There are two main approaches to modelling human behaviour [Domb]. Human Behaviour Processes (HBP) is concerned with modelling internal thought processes and Human Behaviour Output (HBO) is trying to find a correct presentation of the output of thought processes. As will be seen HBO can be used to create realistic behaviour based on experimental data. However, us-

ing a HBP approach would both create realistic output as well as provide a better understanding of how and why the output is generated.

From the definition above, human behaviour can be seen as a change from one state to another, bodily and/or mentally. How the state transformations are performed could be described in interdisciplinary modules based on social, psychological, philosophical, cultural or organizational sciences. The human behaviour model that was implemented here only incorporated psychological phenomena because of the nature of the task.

A computational model of a human [DBDH<sup>+</sup>01] [SBL00] [NLE98] should consist of some input function that interprets the surrounding environment and adds the interpreted data to the memory of the agent. Then there should be cognitive processes that may still further interpret the data and reason about it. Reasoning might consist of problem solving and look-ahead searches. The cognition system also decides what possible actions to take and relays these actions to a motor system. Finally, the motor systems interact with the world and might change the world to a new state.

## 3.2 Cognitive architectures

### 3.2.1 Introduction

A cognitive architecture is an instrument for creating computational models of human behaviour founded by theories of human cognition. The theories and mechanism presented here are all founded on the rationalistic view of cognition. Cognitive architectures<sup>1</sup> are similar to common development environments where users can write and run behavioural models of their own. The architecture provides a language for writing these models and flow control mechanisms. Unlike implementations in traditional programming languages, an implementation from a cognitive architecture is designed to be psychologically plausible. The architectures include predictions concerning parameters and constraints on human cognition, perceptual and motor behaviour. By using a cognitive architecture, the model automatically abides by these theoretical constraints and thus prove psychologically more correct. Typically, a cognitive architecture has different mechanisms for how humans store, forget and recall knowledge. Other constraints might include visual and auditory encoding as well as motor behaviour such as moving an arm. These mechanisms are often designed after studies on physical and psychological phenomena.

There are a wide variety of theories on human cognition and different cognitive architectures that adopt these theories. There are architectures that have simple behaviour mechanisms whereas others use complex computational models for different mechanisms. As architectures tend to focus

---

<sup>1</sup>e.g. Soar, Act-R, Cognet, etc

on different mechanisms, the choice of architecture highly depends on the field of study. For example, an agent might have a very complex mechanism for its visual system and a poor model for the internal reasoning. Sometimes hybrid systems are created that use the "best" mechanisms from each sub-architecture [Cho01]. In addition, it is difficult to separate cognitive architectures from architectures that do not claim to be founded on a theory of cognition. These "non-cognitive" architectures are merely tools for creating behaviour but tend to have similar mechanisms as the cognitive architectures because it is the most appropriate way to represent behaviour. Cognitively plausible or not, most architectures have some common components that can be deduced from humans or animals.

First, one or several input functions interpret data from the surrounding environment. Typically, these functions are called sensors because they sense the environment. The sensors could correspond to human ears, eyes, etc. The sensors then interpret objects and their attributes in the environment to internal abstract representations. These representations are stored in a working memory [YL]. Working memory, sometimes referred to as fact memory or fact knowledge, is a known term in psychology and the definitions may vary. At least working memory can be seen as storage for abstract representations of objects that an entity can reason about. For example, when looking at a car, the eye receives an image of the car, processes that image and sends it to the working memory. Later, that car can still be reasoned about because there is some internal representation of the attributes of that car in memory. Working memory is a collection of abstractions that a mind can reason about and thus represents the internal state of an entity. Many theories put further constraints on working memory to limit the amount of information that can be processed.

As the environment and internal state change, so does the working memory and is therefore sometimes called short-term memory. Working memory does not change only because of changes in the sensor system; internal reasoning such as planning and conclusions will also produce abstractions and changes to working memory.

Working memory cannot stand alone as representation of artificial behaviour because it is only storage of abstractions. A human or agent must also be able to reason about the elements in working memory. How this reasoning executes is stored in a production or procedural memory. Many cognitive architectures use production rules to represent this knowledge and are therefore classified as production system architectures. An example of a non-production system is a neural net system. A production rule is just a condition-action rule that corresponds to if-then clauses in traditional programming languages. There is a condition, or an "if-part", that test one or several elements in the working memory. If the condition is satisfied then the rule "fires" and produces an action defined in the "then-part". The production memory can be used for internal reasoning as well as producing

actions that may modify the environment. If, for example, working memory has two numbers and there is a rule that can add those numbers, that rule will fire and maybe create a new working memory element with the sum of those numbers.

Production memory can be seen as the program that drives the entity forward by reasoning, while working memory only holds variables in form of objects with attributes and values. The production rule can add, remove or change working memory elements and are general so that they persist even if the current situation changes. For example, even if a door is not present at the moment, there is still knowledge how to open a door.

To be able to act in the environment the architecture must also provide one or several output functions. The output function corresponds to motor actions such as producing sounds, moving an arm, turn the head, etc. Motor commands are often triggered by changes to certain working memory elements. The architecture usually executes these functions in a sequential manner: Input, Reasoning, Output. Each such run is called a cycle and are repeatedly executed with some time interval. In the addition example above, the numbers could come from sensor information as if the agent read the values from a book. The sum could be put in the motor-system to write down or vocally express that number.

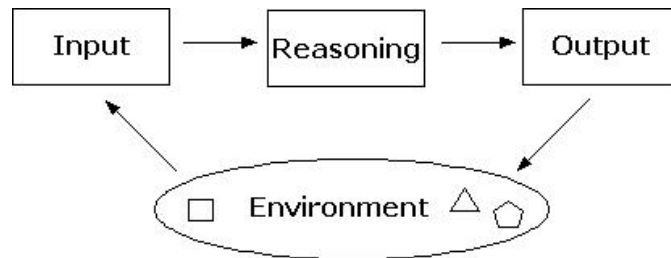


Figure 3.1: A typical cycle in a cognitive architecture

Many theories of human cognition use "goal orientation" as the propelling force that produces behaviour [JERTa]. The theory states that humans are constantly trying to attain goals. These goals can be divided into sub-goals that upon completion have taken the individual one step closer to a higher-level goal. Creation of sub-goals is connected to behaviour and is typically controlled by the production memory or automatically by the architecture. From a traditional programming view, sub-goals can be seen as a way to modularize the program at design time. Depending on the system the sub-goals can also be a way to minimize the current rule-set for the decision algorithm at runtime.

### 3.2.2 Soar

Soar [LLR] [LCC99] [Lai01] [Rit01] [Lew99] is a production system architecture that has been in use since 1983, first introduced by Allen Newell, John Laird and Paul Rosenbloom. The Soar developers say that the program could be seen as either a tool for creating intelligent agents, a unified theory of cognition (UTC) or an implementation of that theory. Newell made an effort to unify so called micro-theories of cognitive behaviour from many different areas [New90]. The result was a unified theory of what cognitive behaviours have in common. Soar has computational models of the mechanisms and structures in the theory. However, as a computer program some mechanisms, not defined in the theory are also implemented. It is possible to model agents in Soar without regard to its theoretical background, although it may be convenient to know that there actually is a theory of cognition that reflects the behaviour of the modelled agents. One of Soar's largest achievements was its use in STOW'97<sup>2</sup> [JLN<sup>+</sup>99] where Soar agents were flying simulated aircrafts in a synthetic military environment. Soar was used in a system called TacAir-Soar<sup>3</sup> where agents were able to fly many different missions and work in teams. How well the Soar theory stands as candidate UTC in comparison to other theories is not within the scope of this project. Soar was chosen primarily because of its availability and because of similar work like TacAir-Soar.

Soar is a system with a working memory, a production memory and a preference memory. Working memory consists of symbols representing objects and attributes that describe the internal state of the agent and what the agent is currently reasoning about. The objects are represented by identifiers that have attributes with attribute values. Attributes can also consist of objects and thus form a tree of symbols. Soar uses goal-driven behaviour where each goal is represented as a state. At any time, the agent is currently working in one of those states to achieve an associated goal. Every state is also an object in working memory, which means that the agent may access this state information and reason about its current goal. When the Soar agent is first started, it is working in the top-state. All elements in working memory must be connected to the top-state identifier. For example, figure 3.2 shows a schematic view of the working memory of an agent. The agent reasons about two cars: a red Porsche and a blue car whose brand cannot be determined.

By using a single representation of all objects, the working memory becomes very dynamic. It is also possible to reason about unknown attributes.

Soar's production memory consists of production rules with a condition-part and an action-part. The condition-part can consist of one or more conditions. If all conditions satisfy working memory elements the rule will be

---

<sup>2</sup>Synthetic Theater Of War

<sup>3</sup><http://www.soartech.com>

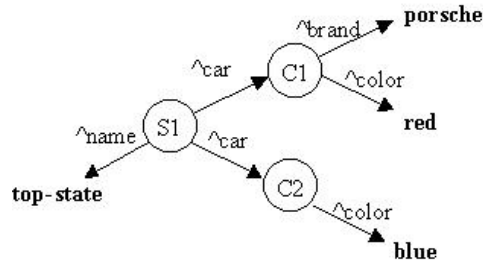


Figure 3.2: A schematic view of Soar’s working memory. An agent, working in its top state, is reasoning about two cars. The abstractions in memory corresponds to a red Porsche and a blue car of an unknown brand.

instantiated and the action part will be executed. Matching working memory elements in Soar is done by placeholders or variables that are substituted by the correct values in working memory.

Soar syntax:	English syntax:
<code>sp {print-all-cars</code>	Name:print-all-cars
<code>(state &lt;s&gt; ^car.brand &lt;brand&gt;</code>	IF there’s a car with a brand
<code>-&gt;</code>	THEN
<code>(write &lt;brand&gt;)} }</code>	print the brand of that car

Figure 3.3: An example of a production rule in Soar. The rule is instantiated as soon as all of its conditions are satisfied. The placeholder (-brand-) will be replaced with the value “porsche” in working memory. If there are several sets of elements that satisfy the conditions, a rule instantiation will be created for each matching set. In this example, a rule instantiation will fire for every car with a known brand.

The action-part can also consist of several actions that most often will add or remove elements in working memory. Soar differentiates between ordinary rules and another type called operators. Actions in an operator rule make persistent changes to working memory and are often used when the agent should perform something. Actions in an ordinary rule are retracted when the condition part no longer matches. For example, an action in ordinary rule may create new elements in working memory. When the condition part no longer is satisfied the action is retracted and the elements are removed. Actions in an operator are not retracted even if the conditions are no longer satisfied.

Operators are executed in three different steps. First, an operator must be proposed in any state. When the proposal is done, a preference is also

attached to the operator. Preferences are relative measures of how appropriate the operator is. It is thus possible to say that one operator is better or worse than the other and if that operator is the best or the worst in the current state. Second, if there are several proposed operators only one of them can be chosen. Soar automatically selects an operator by comparing their preferences. Third, the operator must be applied so that the actions are executed. At first, it seems like the first two steps are redundant but they are provided for an automatic sub-goal mechanism. When the Soar architecture do not know how to proceed it reaches an impasse. When this happens Soar automatically creates a new sub-state. The creation of a sub-state is a signal that there is a lack of knowledge and the new goal represented by the sub-state is to obtain that knowledge. A new state is created when:

1. Two or more operators are proposed and Soar cannot decide which one to choose.
2. An operator is proposed but there is no rule that applies the operator.
3. There is not any operator proposed.

The first case emerges when there is a preference conflict. Two or more operators might be equally appropriate. This situation is often used as a way of doing problem solving; a sub-goal is created to find out which of the proposed operators is the most appropriate one.

The second case is often used when the operator cannot be expressed by a simple action. For example, an operator for patrolling might actually be expressed as several operators or actions and therefore a single operator cannot be applied. A new sub-state is created called patrol where these sub-operators can be applied. This provides a way to break down high-level operators into several low-level operators similar to the techniques used in HTA (see chapter 2.5).

The third case is often used to catch idle activity when the current situation causes the agent to simply wait.

An impasse is resolved when the conditions that once created it are no longer satisfied. When this happens Soar automatically removes the sub-states, because the intention of the sub-states was to resolve this impasse. Impasses are not only resolved by changes in sub-states. In addition, external events from the surrounding environment may change the internal state of the agent abolishing the impasse. Solving impasses are also used in Soar's learning mechanism which was not used in this study.

### 3.2.3 Agent Factory

Agent Factory [AJ] [AL] [AJL] is a production rule architecture developed by Pitch Kunsapsutveckling AB<sup>4</sup> in Linköping, Sweden. In contrast to

---

<sup>4</sup><http://www.pitch.se>

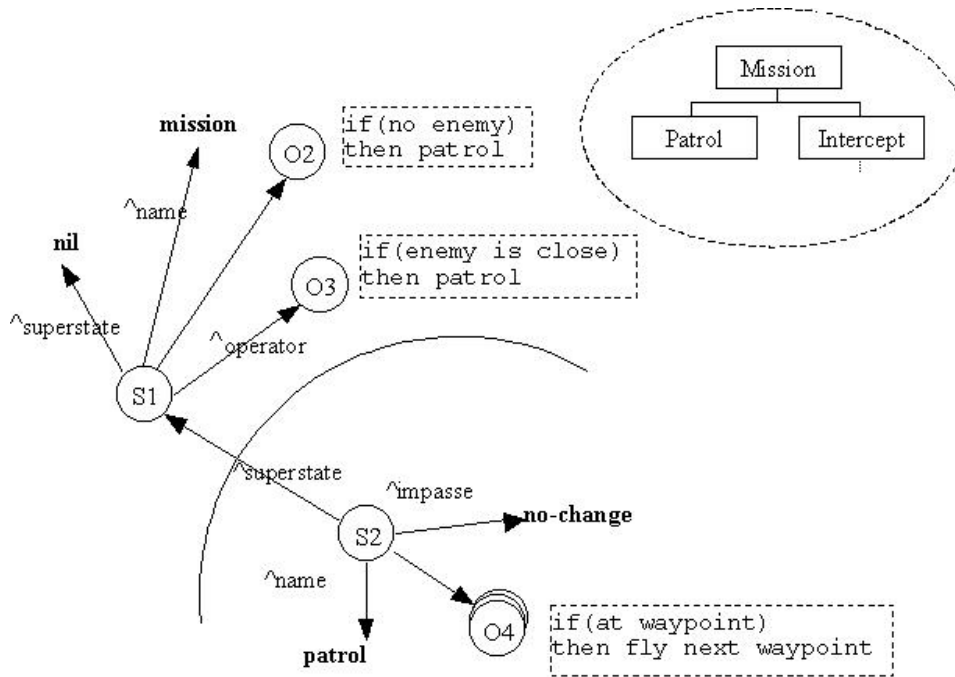


Figure 3.4: A schematic view of the working memory and representation of states in Soar. The figure illustrates two mechanisms. The high-level operator patrol is chosen because there is no enemy in sight. Because no operator application exists a new sub-state is created called patrol. In that sub-state new operator can be proposed that are sub-operators of the patrol operator. The purpose of these sub-operators are to attain the patrol goal by flying through way-points. If the agent is conducting a patrol and an enemy suddenly appears, the conditions for proposing the patrol operator are no longer satisfied and the sub-state patrol will be removed. Also, the intercept operator will be proposed, since the condition for its proposal now matches.

Soar, Agent Factory is not a realisation of a theory of cognition. Instead, the architecture has been developed merely as a tool for creating intelligent agents. Because of a rather united view of modelling behaviour, the Agent Factory architecture has similar mechanisms as Soar and other cognitive architectures. As a modelling tool, one of the main ideas behind Agent Factory is to abstract modelling to a level where experts can easily express behaviour representation without any programming skills. The generated agents consist of Java code and are HLA<sup>5</sup> compliant. Agent Factory uses HLA to weld components together into an agent system.

HLA can be seen as a framework for distributed simulations. Entities and components that want to participate in a simulation are called federates that run their own simulation systems. To exchange information with other federates a part of the internal system can be made visible to other entities by publishing some of the objects and attributes. For example, an aircraft might publish its position and altitude so that other entities can see it. It need not publish objects that are only used in the internal system. To know about other entities a federate subscribes to relevant information. Ground artillery federates could, for example, subscribe to position and altitudes published by aircraft federates. A device called RTI<sup>6</sup>, handles all communication between federates. The components that are created need not represent separate platforms, an aircraft can consist of many federates that represent system components such as radar, weapons and joystick. This component-based approach makes it easy to build platforms from libraries of system components.

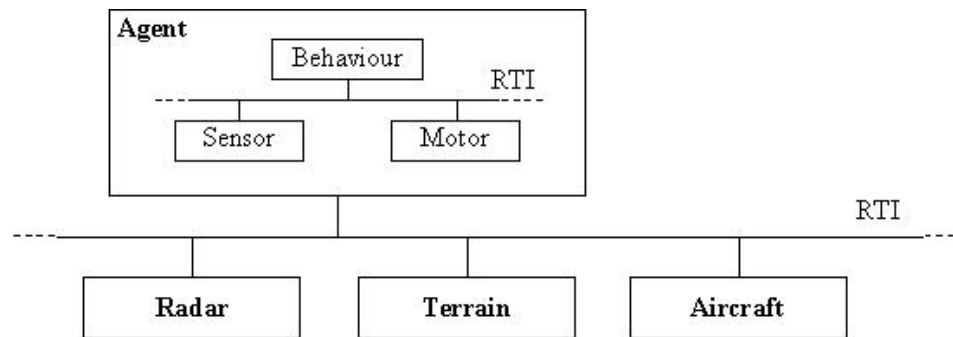


Figure 3.5: The RTI structure in Agent Factory.

Typically, the agents consist of a sensor system, a motor system and a behaviour system that are all connected to an internal RTI forming an internal federation (see figure 3.5). The sensor system is supposed to interpret stimuli from the external federation or outside world by subscribing to rel-

<sup>5</sup>High Level Architecture

<sup>6</sup>RunTime Infrastructure

event information. After necessary processing at the sensor component it then publishes information in the internal federation so that the behaviour component and then working memory is updated. The behaviour component applies the reasoning and finally causes the agent to take some actions. These actions are published and relayed to the motor system. After the motor system has processed the actions, it publishes information to the external federation. As in the external federation, the internal federation can consist of several components. There might be a few sensor components and a few motor components that all can communicate with each other internally producing a more complex system.

All objects and attributes that an agent should be able to reason about must be predefined in an object oriented conceptual model. The model is created with an Object Modelling Tool (OMT) where objects are presented as classes with attributes. Objects that are not introduced in the conceptual model cannot appear in any reasoning by the agent. In an agent editor tool (see figure 3.6), some of the defined objects can be assigned as either sensor or motor components and one object is defined as the agent. This step automatically creates subscriptions and publications between the behaviour system and the motor and sensor system. When the sensor system publishes information that the behaviour system subscribes to that information is automatically added to the working memory without any need of structuring that information any further.

As changes are made to the working memory there is a mechanism for elaborating these data further. The elaborations are written as rules that fire as soon as the condition part matches elements in working memory. This "common-knowledge" rule set is not associated with any particular goal and thus fire independently of the current situation.

A rule consists of a condition part and a conclusion part. In Agent Factory, conditions and conclusions are written separately so that these rule components can be used in different rules. Rule components are written as Java functions and assigned appropriate names. There is also an option to associate a documentation field with the components. This enables experts without any programming skills to create definitions of rule components. A programmer may then look at the definition and the documentation to implement the component with Java code.

The mechanism for modelling agent behaviour uses a goal-oriented approach. Each goal can consist of a number of sub-goals. A number of plans can be associated with each sub-goal and each plan consists of a number of phases (see figure 3.7). Every goal and plan has an associated condition that specifies whether the goal or plan should be chosen. All these conditions, called reactive rules, guide the behaviour and the traversal of the tree. Phases are the nodes of the tree and are executed serially. Typically, each plan has several phases. Each phase has a terminating condition that specifies when the phase is done. If the terminating conditions are satisfied the

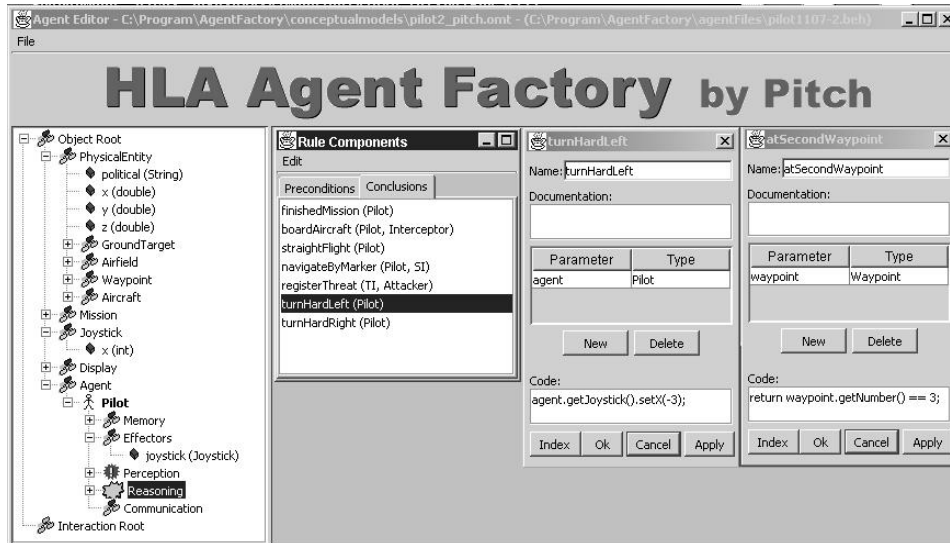


Figure 3.6: Agent editor tool in Agent Factory. The left frame holds a tree of all objects that the agent can reason about. The pilot object is set as the agent itself. The agent automatically receives memory, effectors, perception and a reasoning structure. The effectors structure consists of objects that the agent should be able to affect, in this case; a joystick. The perception structure consists of general rules that fire when its conditions are satisfied, independently of the current situation. The left window of the right frame consists of conditions and conclusion components. The middle window shows a more detailed description of a conclusion component and the right window shows a condition component.

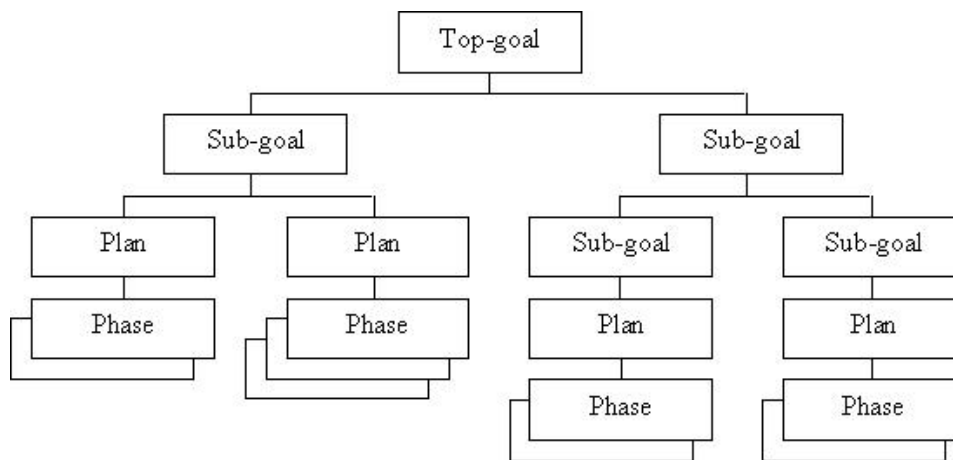


Figure 3.7: Goal-Plan-Phase tree in Agent Factory.

next phase is chosen. Phases also have an associated action that should be performed when the phase is executing. Actions are specified by conclusion components. Condition and terminating conditions are specified by condition components. Adding new goals, plans and phases is done graphically by editing the tree and by dragging rule components.

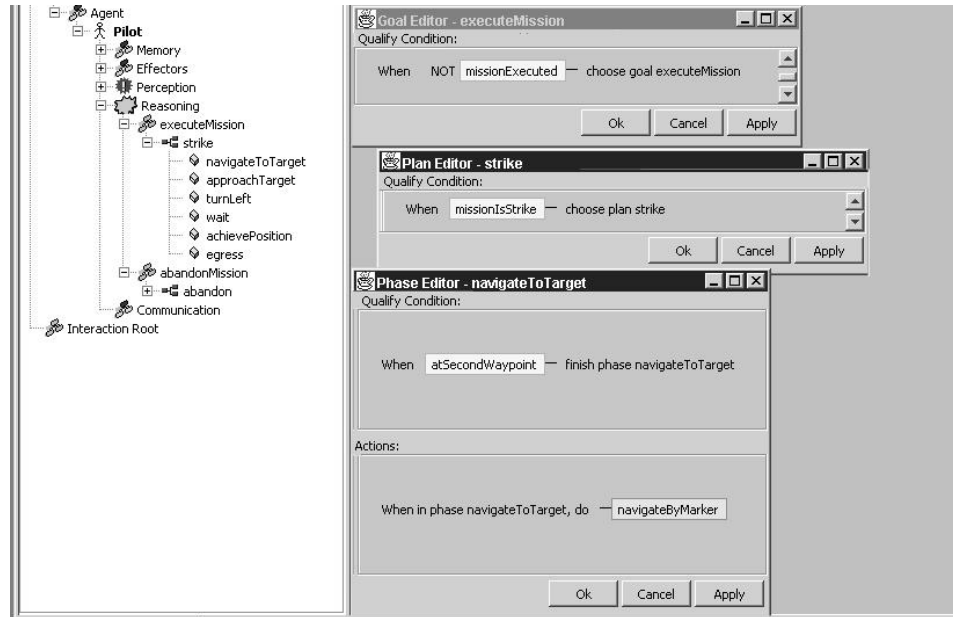


Figure 3.8: Reasoning in Agent Factory. The reasoning structure consists of goals, plans and phases (left frame). A goal is chosen when its conditions are satisfied. Conditions are created by dragging condition components to the goal editor (right frame, top window). A plan is also chosen when its conditions are satisfied (right frame, middle window). A phase however, has an end condition and an action (right frame, bottom window). When the end condition is satisfied, the phase is completed and the next phase is chosen. During the execution of a phase, its associated action is executed.

During a decision cycle the agent receives data from the sensor systems. If there are any general rules that can fire, they all do so. The firing may cause additional rules to fire until quiescence is achieved. After the general rules have fired, the reactive rules will traverse the goal-plan-phase tree until the topical phase is reached. By looking at the terminating conditions of the phases, the architecture decides which phase that should execute. When that phase is executed, its associated conclusion component will fire. If objects that are assigned as motor components have been modified during the cycle, they will be published and received by the motor system. This cycle then repeats. Note that the goal-tree does not have any state memory.

The goal-tree is traversed from the top-state in every cycle.

### 3.2.4 Comparison

Because different architectures have different mechanisms and features, the choice of architecture must reflect the current modelling task. The architectures studied here seem to have similar mechanisms. From a modellers perspective a comparison was done, in order to highlight differences and some useful features. Performance issues such as speed, memory usage, etc has not been considered. The comparison has been divided into different parts reflecting different architectural mechanisms.

**Situatedness** The term is used to describe the possibilities of deploying agents into environments and situations. That is, how communication is handled between agents and environments.

*Agent Factory:* As Agent Factory agents resides on the HLA architecture the communication between the agent and the environment is done automatically by the RTI. All information that is supposed to be exchanged is predefined in a conceptual model. There is no need to translate this information to a structure consisting of working memory elements. Working memory is thus updated automatically. If filtering is needed sensor- and motor-system components could be added to the RTI. For instance, one module could be used to handle visual filtering, one module for auditory filtering and one module for motor commands. In addition, these modules could communicate and affect each other to create more complex filtering systems. This component-based architecture provides a high degree of re-usability.

*Soar:* Soar must be provided with an external module that communicates with the environment. That module must receive objects and attributes from the environment and translate those objects into a working memory structure, a process that could be very complex for large systems. Also, changes that the agent makes to the environment must be translated and sent to the environment. For large sets of input and/or output, this could be a challenging task although the Soar architecture provides some functionality to create such modules by C or Tcl code. [CS95] There are also third party packages available to simplify communication and structuring. In some cases, the behaviour must also be extended with rules that monitor the output system.

**Representation of knowledge** Artificial knowledge can be represented in different ways, this section compares two different representations of working memory.

*Agent Factory:* In Agent Factory working memory is represented as objects with attributes. All objects that the agent should be able to reason about must be predefined by creating classes and attributes in a conceptual model. Working memory is thus static so that new classes or types of elements cannot be added or removed. If objects appear that are not in the definition it will cause the agent to crash. The agent cannot reason about unknown objects or attributes. In addition, all internal reasoning and abstractions must be defined as objects or attributes. Internal reasoning often creates temporary abstractions that are only meaningful in the current situation. For example, during an air combat situation the agent may create abstractions such as "distance is sufficient", "little low", "angle of attack OK". The agent will then reason about these abstractions and not the actual numbers. If such abstractions are to be used, they all need to be defined in the conceptual model and they will exist throughout the whole lifetime of the agent. For complex behaviour, many internal abstractions may be needed. This would lead to difficulties to manage and update the conceptual model. In addition, agents cannot reason about its current state or previous states. To achieve this, additional objects must be introduced in the conceptual model.

*Soar:* A single representation, namely symbols, is used to define working memory. Symbols are connected to form a tree of objects and attributes. The symbol tree is very dynamic and need not be predefined. The agent may add, remove and change elements in the memory structure and connect these symbols in arbitrary ways. In addition, the current goal or operator is part of the working memory, which enables reasoning about the current state and previous states. The symbols are either modified by internal reasoning by the agent or a communication module. The agent often has a sub-tree of symbols destined to receive perceptions from the environment, these symbols are often structured and modified by the input module.

**Representation of behaviour** This section looks at the flow-control mechanisms that produce behaviour i.e. production memory or long-term memory.

*Agent Factory:* Long-term memory or the reasoning consists of separate condition and conclusion components. The components could be used in general rules that fires as soon as working memory elements have changed and satisfy the condition part. The goal-plan-phase tree sets up the goal-oriented behaviour where condition components aid in the traversal of the tree. The goal-plan-phase tree is static so that goals, plans and phases cannot be reused in different parts of the tree. They must be redefined at all occurrences.

*Soar:* Long-term memory consists of production rules that are catego-

rized as ordinary (I-supported) rules and operators (O-supported). Since Soar uses an automatic sub-goal mechanism that dynamically chooses a goal, it may not be that easy to understand or to predict behaviour. The mechanism provides however, a very dynamic behaviour where several operators can be evaluated in look-ahead searches. Soar, does also have an option to randomly choose a goal when both operators are equally favourable. The automatic creation of sub-goals makes it possible to reuse operators and sub-goals in several high-level goals. This provides the use of libraries with complex operators that can be called upon in different situations (see figure 3.9) For example, a complex manoeuvre "zig-zag" that consists of several sub-operators might be used in different situations when avoiding missiles, artillery or radar-lock. The capability of planning [JWvLL] and learning might also be useful in future developments of complex behaviour.

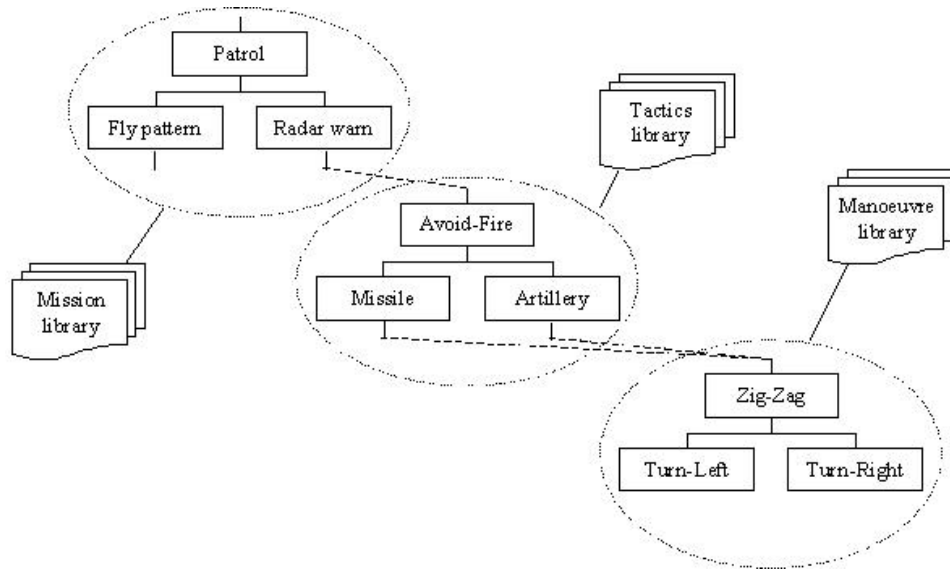


Figure 3.9: Reusing states in libraries.

**Interface** Here, the architectures are reviewed as tools for creating behaviour. The interface toward the user is investigated.

*Agent Factory:* Agent Factory provides a graphical interface for the modeller. Condition- and conclusion components are assigned by their names and have an optional documentation field that makes it easy for non-programmers to use them. There is also a graphical representation of the goal-plan-phase tree and since the tree is static, it is easy to predict the path and the resulting behaviour.

	Agent Factory	Soar
Situatedness	+	-
Knowledge	0	+
Behaviour	-	+
Interface	+	-
Community	-	+

Table 3.1: A brief comparison.

*Soar*: Soar has its own syntax for writing rules that may not be so easy to understand for non-programmers. There is no graphical representation of behaviour or working memory. Since both representations are very dynamic, it may prove hard to predict behaviour even for Soar programmers.

**Miscellaneous** Other factors that may be of interest.

*Agent Factory* During this study, Agent Factory was undergoing some architectural changes for a new version. The new version will have some new features, but the comments on the evaluation will still hold for that version. Agent Factory is a relatively new product with a very limited set of users.

*Soar* Soar has been in use since 1980 and has gone through several improvements and architectural changes. There is a small, but devoted community using Soar for various applications that is involved in this continuous refinement.

### 3.3 Human psychology

Whereas cognitive architectures or theories explain some psychological phenomena of processing sensor information, motor commands and reasoning they seldom incorporate mental models of mood, feelings, stress etc [Tac97] [RA00]. Integration of such mental models is very difficult to realize because cognitive architectures are general tools while mental processing is often very dependent on the task and on individual characteristics. In addition, these models are often based on observations in real life making them even more situation dependent. This is one of the criticisms to the rationalistic tradition of cognitive science.

#### 3.3.1 Psychological models

Human behaviour depends very much on the current situation, current task, current internal state and the current mental state. Mental state can be the result of social, psychological, philosophical characteristics and influences that they may have on each other. Because psychology represents one such module, and perhaps the most influent one, there is a demand to analyse

how such phenomena affect behaviour. However, psychological factors such as stress and fear are very hard to portray and impossible to quantify. There are, however, means to estimate such indices. Psycho-physiological factors consist of attributes that are measurable and often controlled by psychological factors. For example, heart rate is a typical psycho-physiological indicia. Heart rate is very hard to actively change. However, when humans become nervous or frightened heart rate increases. It may then be possible to estimate if the person is nervous or scared by studying his heart rate.

In order to create human behaviour that is more complete, it is desirable to create models that describe the relationships between psychological, psycho-physiological and physical indices. Several models were created after studies on military pilots conducting both real and simulated missions [SF99] [SAT<sup>+</sup>92]. The model was constructed by means of structural equation modelling with LISREL<sup>7</sup>. LISREL allows creations of structural casual relationships between variables whose dependencies might not be obvious or well defined. In the field of psychology, many hypotheses of how the human mind works involves experimentation with unobserved variables and how they are affected by observed data (stress and heart rate). Path analysis was applied to test the model fit and statistical relations. The psychological unobservable variables that were introduced were based on subjective estimates as well as objective assessments. Post mission analyses and interviews produced estimates of mission difficulty, situational awareness, perceived performance, concentration, motivation, control, vigilance, mental capacity, mental and physical effort, concentration, information load (TSD, TGD), priority to tasks and complexity of information. Hypothesised in the model are casual structures among unobservable factors measured by observed variables.

The studies were performed on pilots conducting strike mission with high speed and low altitude, which was one of the reasons such a scenario was created in this study. The objective was to manoeuvre the aircraft while attending to the current tactical situation. These tasks are typical for a military pilot and often they tend to compete for mental resources. For instance, flying at a low altitude requires attention to the HUD while tactical situation assessment requires attention to the HDD's. The average pilot in the study did not prioritise any of the tasks at the expense of the other. Not only may the resulting model be appreciated but observations and their relationships that the model was created from might also be useful. Before introducing the model, a few relationships between these observed parameters will be presented. An interesting point is that psychology has long been faced with the so-called magical number seven[Mil56].  $7 \pm 2$  is frequently obtained as a threshold value when studying humans ability to discriminate between several objects. This phenomenon of working memory limitations can be applied to all senses and also appeared in theses studies.

---

<sup>7</sup><http://www.ssicentral.com/lisrel/word.htm>

**TSD - Altitude** The ability to maintain the instructed altitude gets worse as the number of objects on the TSD increases. Both altitude and variation in altitude increases and the corrections of altitude errors are delayed. Studies have shown that experienced pilots can maintain their altitude when the number of objects on the TSD is lower than 10. For less experienced pilots the number is eight. When the number of objects has reached that value, the experienced pilots ascend 1m per additional object on the TSD. Less experienced pilots, ascend 2m. Of, course these numbers also depend on the topology of the terrain. Even so, the deviation between current and instructed altitude is highly dependent on the information complexity on the TSD.

**TSD - TSD reports** The number of reported objects is dependent on, and not necessarily the same as, the number of objects presented on the TSD. By average, the pilots lost track of one out of five additional objects. This tendency started somewhere between 7 to 11 objects. The ability to perceive changes declined and the number of objects was underestimated as the number of objects increased.

**TSD - Heart rate, eye fixations** During the studies, variations in heart rate frequency went up and down in phase with the information complexity on the TSD. Also, the fixation times and fixation change frequency changed as the complexity increased.

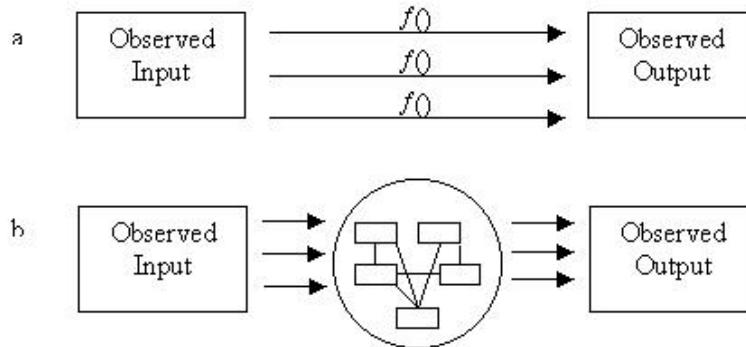


Figure 3.10: a) Every element of observed data has a direct relation to a particular element of behaviour. b) All elements of observed data are collected in the psychological model. The model consists of causal relationships between psychological mediating indices. Some of these indices also affect behaviour.

By using these relationships from observed data more realistic behaviour can be modelled. Some sets of input results in some sets of output. These

relationships could be reflected in an agent and the psychological model would be automatically inherent. There are however advantages in using computational psychological models. Without the model, indices such as information complexity on the TSD would directly influence heart rate. This assumption is not very realistic. Intuitively, there should be one or several psychological mediating factors that influence behaviour, psychological and psycho-physiological indices. For instance, as information complexity increases a stress level factor might be affected and stress is known to affect heart rate. In addition, if several psychological factors are introduced, they may affect each other. Stress and fear could for example influence each other producing different behaviour. These relationships cannot be used if the psychological model was to be bypassed.

The mediating factors are psychological measures that are constructed based on subjective and objective statements as well as physical and psycho-physiological measures. The study introduced pilot mental workload (PMWL), General Workload (BFRS), Mental Capacity reduction (CAPAC), Situational Awareness (SA), Motivation (MOTIV), Pilot Performance (PP) and Mental Effort (EFF) as psychological factors. Many of these factors are common terms in psychology and they also have associated methods of assessments. Mental workload, for example, is often assessed by a method called NASA-TLX<sup>8</sup> and the Bedford Rating Scale is often used for assessments on situational awareness. Model analyses showed several relationships, for instance the complexity or difficulty of the mission affects the mental workload of the pilot. Mental workload, in its turn, affected different aspects of performance. This result correlates well with the well-known phenomena that it is often the mission requirements themselves that are the primary sources of stress [Tac97].

Even if the model in figure 3.11 has been justified and to a high degree proved correct by complex methods and large sets of data, it is still not useful as it is for computational modelling. How should the psychological indices affect physical behaviour? For example, a pilot with poor performance might have larger reaction times and a low degree of situational awareness could result in misjudgements. The lack of relationships between the psychological indices and observed behaviour call for an extended model. At FOI such models exist and very large sets of experimental data that will provide extension possibilities. Unfortunately, the models and the data were unavailable during this study. An example of a model from an earlier study is shown in figure 3.12. The model has in fact incorporated factors with more detailed descriptions of the resulting behaviour.

---

<sup>8</sup>NASA Task-Load Index

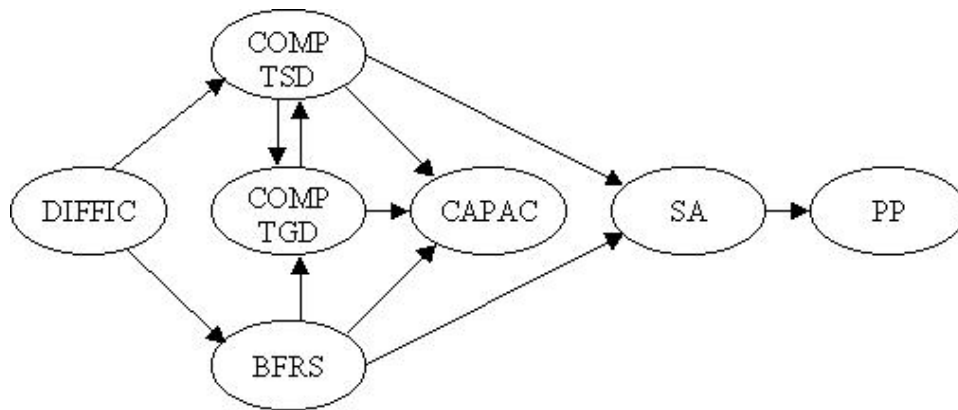


Figure 3.11: A structural model of causal psychological relationships created with LISREL. The model has its root in the factor representing the difficulty of the mission which have positive effects on complexity (COMP TSD) and mental workload (BFRS).

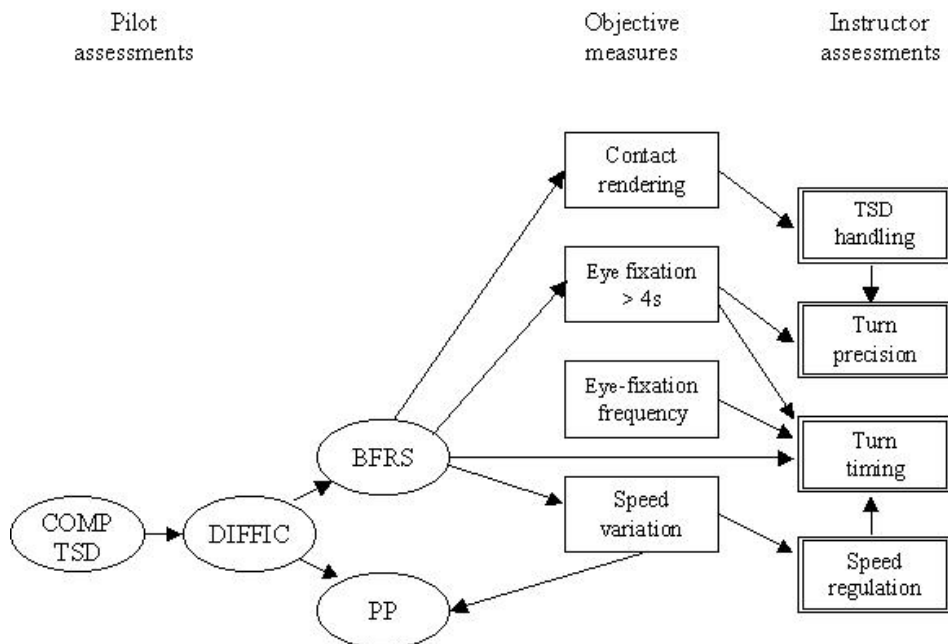


Figure 3.12: A structural model of causal psychological relationships created with LISREL. The model has its root in the factor representing complexity on the TSD (COMP TSD). Some of the nodes describes performance modifiers that could be used to create more complete models of behaviour.

### 3.3.2 Attention

*“It is the taking possession of the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought”*

William James  
The principle of psychology, 1890

For both humans and agents acting in an environment, attention is an essential part of the cognitive system. The cognitive system needs objects with attributes such as colour, smell and sound characteristics; to reason about the external world. Humans have a very limited capability of processing simultaneous information while at some times a vast amount of new information might be available. In complex decision-making systems where agents are supposed to aid in processing vast amount of data, the only limit is set by the underlying hardware and software. Agents that are supposed to behave like humans should however be further constrained and inherit human-like limitations on attentive processing. As with humans, agents should not be fully aware of the current situation at all times. To be able to process some of all the available information without being overloaded and blocked, a filtering process occurs. This process is called attention or focus of attention and its primary purpose is to filter out excessive information. As James suggested attention may be seen as a careful allocation of scarce resources. In literature, attention is often divided into subcategories such as:

**Selective attention.** A process of selectively allocating resources to objects. For example, a pilot might choose to focus on the altitude meter when he is landing. This kind of attention is limited to a few stimulus attributes at one time.

**Focused attention.** A process where some, less important, processes are, rejected for other processes, For example, a pilot ignores clouds and birds when he is scanning the sky for hostile aircrafts. The inability to reject irrelevant processes will make it more difficult to process the relevant information.

**Divided attention.** A process of sharing resources among several processes simultaneously. For example, a pilot must sometimes process information given to them by radio communication while they are executing a complex manoeuvre forcing them to watch the angle of attack metre.

**Sustained attention.** The capability of maintaining focus over an extended period of time. Concentration difficulties make it hard to be focused on an object for a long time.

### 3.3.3 Visual attention

Visual attention [Hum98] [Hil] is the filtering of objects that appear in the visual system. Because there is too much information in the visual field that overloads the cognitive system, a focus that filters out excessive information is needed. In this study, perception is done only through a simulated visual system. There are many theories and models within the field of visual attention. Many of them have different ideas of how and where the filtering is done. Some theories posit that blocking occur at pre-attentive stages and some say that all visual information reach the working memory but is selectively forgotten and removed. Visual attention is frequently compared to a spotlight; it is limited to a particular region constrained by the visual field and can be moved freely between objects. Note that this has nothing to do with the eyes; the eyes only create a limit to the possible region of attention. Thus, visual attention is not tied to the fovea; it can be devoted to other parts of the visual field. In addition, the visual attention may change while the eye-point of gaze remains at the same spot. It is also possible to attend to two objects at the same time. This type of visual orienting is called covert, whilst in overt orienting eyes or even head or body are moved to shift attention to other objects. Covert attention is very difficult to model based on experimental data on how humans move their eyes and therefore most implemented theories use overt attention as a substitute for visual attention.

A cognitive architecture that uses a theory of visual attention is EPIC [Cho01]. EPIC has separate sensor processors, such as visual, ocular and auditory processors. It has also processors for controlling motor commands such as vocal and manual motor commands. EPIC's perceptual visual processor uses four concentric retinal zones:

**Bouquet.** The very centre of the retina with a radius of 0.25 of visual angle.

**Fovea.** A region between the radius 0.25 and 1.0 of the visual angle.

**Parafovea.** A region between 1.0 and 7.5 of visual angle

**Periphery** All visual space outside 7.5 of visual angle

Different features of objects are only passed to working memory depending on the location of the object in relation to the different zones. For example, all perceptual features of an object within the bouquet would be passed to working memory while the shape of the object would be perceived only in the regions within the parafovea. EPIC and Soar have been used to model a human air traffic controller where EPIC's visual processing was used. In that example, text was only passed to working memory when inside the bouquet and the colour of aircraft symbols could be perceived in all regions except the periphery.

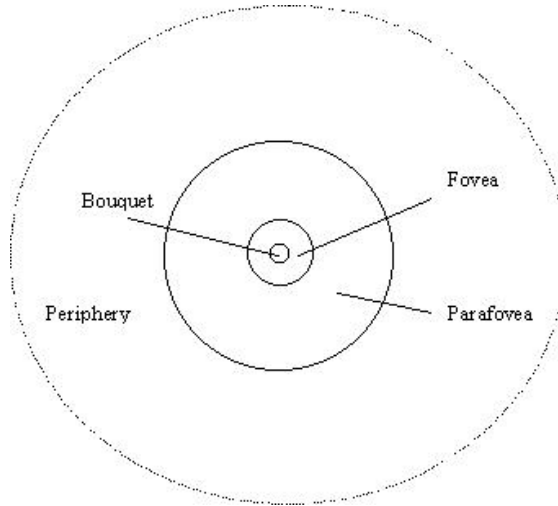


Figure 3.13: Retinal zones in Epic.

Similar to EPIC's constraints on perception of the visual field at close range, the distance to the visual objects will also limit the ability to distinguish object attributes. For example, it's very hard to determine the colour of an object from a long distance. An entity might at some distances be perceived as a vehicle, at some shorter distances be classified as a tank and at close range be classified as a T-80 tank. Instead of a spotlight, a zoom lens metaphor is used with a reciprocal relation between magnification and visual field. This means that a greater field of view reduces the level of detail and vice versa. An ideal lens should produce a constant amount of information across all zoom levels. With this type of filtering, a mechanism for grouping could be applied to reduce the processing further [Hil]. Similar objects could be grouped as one large object to reduce the workload on the agent. At a very low resolution a group is perceived while at higher resolution the group is treated as several individuals. This is very useful in real-time applications where processing of each individual object might drain the resources from other tasks and for example cause a pilot to lose control of manoeuvring the aircraft and crash.

Fixation or dwell time is another aspect to consider when adding object attributes to working memory. It is reasonable to believe that a longer fixation time on one object would result in a more detailed description in working memory. Studies have shown that there is a relationship between fixation duration and recall accuracy [Har]. A study of traffic pilots identified two kinds of fixations, one that serves to monitor instruments and one that uses instrument information for further processing. The first fixation was classified as just a glance, while the other took about 200-300ms longer.

In the EPIC-Soar example above a very high resolution was used to study visual attention with aircraft controllers. As mentioned earlier the need for finer resolution depends on task and also on available data on visual attention. In this study a very simple visual model has been used with several assumptions and simplifications.

**Assumption 1** *The agent must choose to attend to one of four available displays at any time.*

**Assumption 2** *The agent has the capability to percept all information on the currently attended display at any time*

In the model of the aircraft, there are four important displays (see figure 2.3). One of them is the HUD where the pilot can see the sky through a shield of glass as well as information projected on that shield. The other three are HDD's and forces the pilot to look down in the cockpit.

The result of assumption 2 is that the agent's working memory is updated with the new attribute values as soon as the agent attends to the corresponding display. In EPIC, this would correspond to a bouquet with a radius of the current display. This simplified model is just what cockpit display designers want to achieve. Symbols, graphical presentations and filtered data are carefully designed so that the pilot can quickly interpret the current situation. However, based on experimental data this assumption does not hold. Studies have shown that pilots under stress have difficulties in maintaining their instructed altitude. The altitude meter is typically on the HUD. If the assumption would hold this would mean that the pilot never looks at the HUD, which they do. However, EPIC's finer resolution would certainly explain how the pilot attend to maybe more important data on that display. To compensate for the coarse resolution another assumption is made:

**Assumption 3** *A half second is the least time the agent might look at a display. In that time he will perceive all information on the currently attended display.*

When studying pilot's eye movements between HUD and HDD there were two types of eye fixations [SAT<sup>+</sup>92]. In the first type, the pilot seemed to check one of the instruments and the time was 0.5s. In the other type, the instrument contained new essential information and the time was just over one. The time 0.5s is then the time the pilot will need to scan all attributes on the currently attended display.

### 3.3.4 Shifting focus of attention

Another way to classify attention is from the stimulus or objects point of view. The stimulus could attract attention by endogenous or exogenous

methods. Endogenous or top-down controlled methods are directed by the cognitive system and depend on the subject's intentions and actions. For example, reading a line of text from left to right would be a typical endogenous method. Exogenous or bottom-up controlled methods are processes when attention is attracted by external events. These events could be sudden bursts of light, colour, motion, etc. In this case, the subject reacts on stimuli without any connected intentions.

It is very difficult to model shifting focus of attention at a very detailed level. Even though endogenous control is a deliberate act, shifting attention might not be static and a function of time making it hard to predict the exact pattern of attention shifts that the pilot will use. Even the pilots cannot determine in advance at what times and at what regions he will shift his attention to. For example, a pilot flying a route between two points many times, might not produce the same pattern of attention shifts every time. In addition, that pattern will change depending on the internal status of the pilot and the surrounding environment. For example, a pilot would control his heading bug more often when he is undertaking a turn and more seldom when he is flying straight. How should one model the internal mechanism that produces a shift of attention?

**Assumption 4** *At any time, shifting focus of attention is controlled by a random process.*

The random process in this case is a Markov Model [RJ86] [Kle99]. A Markov Model is defined by a number of states and a number of transition probabilities between those states, here referred to as visual states. The visual state reveals which display the pilot is looking at; since there are four displays, there are four possible states. The transition probabilities determine the probabilities of moving from one visual state to another. For example, if in a particular situation, the pilot is looking at the visual indicator the probability that he remains in that visual state might be 80% and the probability that he will shift his attention to the tactical display might be 20%. As mentioned earlier the transition probabilities might change over time as the current situation changes. From studies of traffic pilots one of the conclusions was that the cognitive models should be able to adopt several scan strategies. By studying eye muscle activity for pilots it can be also seen that in different phases of the mission there are different scan patterns. As the control is endogenous and we use a probabilistic method, this translates into assumption 5.

**Assumption 5** *At any time the agent can choose how to distribute the transition probabilities that controls the shift of attention*

The pilot can thus choose which Markov model to use. For example when turning, he might choose a chain that have a very high (90%-100%)

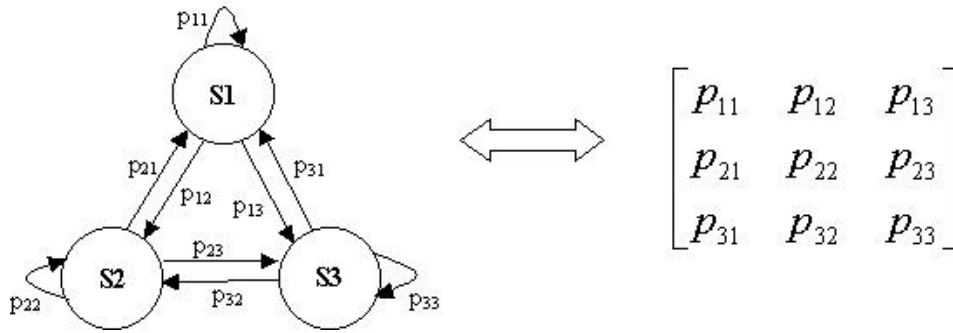


Figure 3.14: Graph & matrix representation of a Markov Model.

probability of remaining in the visual state focusing on the HUD. When the pilot is flying straight he might choose a chain that have equal probabilities of shifting between states, corresponding to a routine scanning of all the displays.

Even though the use of Markov models involve probabilistic processes, they also have a feature of incorporating experimental data into it. Markov chains are often used in pattern recognition where the chain is trained so that its transition probabilities are tuned to match observed data. This provides an opportunity to analyse real data and to capture individual scan characteristics from different pilots. The probabilistic process also creates different behaviour every run. The problem with probabilistic models is that they might produce unrealistic behaviour. For instance, the pilot might "get stuck" in one of the visual states for a too long or a too short period of time. The constraint in assumption 3 avoids the "short-time" problem.

Intuitively the control of shifting attention depends on the information that is important in the current situation and how often that information must be updated. For instance, while flying at very low altitude the altitude meter must be checked regularly, but not necessarily all the time. If the altitude meter has not been checked for a while the focus control mechanism will certainly cause the agent to attend to that metre. To model this mechanism some sort of timer must be attached to each attribute value in working memory. The timer attribute is then checked and if the timer value crosses some threshold value, a new attribute value must be retrieved, causing the agent to attend to the corresponding display.

Exogenous processes capture attention involuntarily when an objects feature or attribute reaches a threshold value. When this happens the sudden onset of that stimuli causes the object or feature to automatically place itself in working memory of the subject. An example would be a sudden flash of light that causes the beholder to react immediately. It is of course possible to use a filter that is controlled by impediment factors such as fatigue causing

the agent to react in other ways.

### 3.3.5 Attention and psycho-physiological indices

Assumption 5 says that the agent has total control of the distributions in the random process controlling shift of attention. Here another assumption is made so that the agent only receives partial control.

**Assumption 6** *The transition probabilities are partially controlled by the pilot and partially dependent on the number of objects on the TSD.*

By studying pilots' eye movements, [SAT<sup>+</sup>92] a correlation was found between fixation times and number of objects on the TSD. In addition, there was a dependency between attention shift frequencies and the number of objects on the TSD. The fixation times on the HDD's grew linearly with the number of objects. The fixation times on the HUD decreased until the number of objects reached six, then the time was approximately constant with a value of about two seconds. It seems that two seconds was the average time that was needed to have a sufficient control of manoeuvring the aircraft. Whether this change in fixation times depended on some internal process that is not connected to any goal or not is difficult to answer. From one of the theories above, the longer fixation times may be the result of limitations on forwarding visual objects to working memory. Here, we use the former explanation. The fixation times depend partly on stress and the number of objects on the TSD. Studies have also shown that less experienced pilots tend to "stare" for longer times and even change their pattern of scanning while they are making conscious decision. Even experienced pilots tend to disrupt their scanning behaviour in these situations but for a shorter amount of time.

If a psychological model was to be used assumption 6 would not be applied. From studies on pilots it is shown that scanning behaviour and fixation times are very much dependent on different aspects of the psychological indicia mental workload.

### 3.3.6 Situational awareness

*"A pilots continuous perception of self and aircraft in relation to the dynamic environments of flight, threats and mission, and the ability to forecast, then execute tasks based on that perception"*

McMillan, G.R.

Report of the Armstrong Laboratory SAINT

There is no agreed definition of situational awareness, the definition above is restricted to pilots. A high degree of situational awareness can be the result

of awareness of personal limitations, the current situation and the possible outcome of the current situation. In this study, the awareness of the current situation is used as a substitute for situational awareness. This type of awareness does also affect the accuracy of predictions concerning possible outcomes.

**Assumption 7** *The number of objects on the TSD causes the agent to lose awareness of his current altitude.*

From studies, it is shown that the number of objects on the TSD causes the pilot to deviate from his instructed altitude. The altitude was maintained as long as the number of objects was seven or less. For every additional object to that limit, the pilot tended to rise 2 m. The measure was 1m for more experienced pilots. It's difficult to explain this phenomena. The explanation given here is that the pilot assumes that he is below the instructed altitude. When the number of objects increases the deviation between current altitude and perceived altitude increases.

## Chapter 4

# Implementation

To better understand the differences between the agent architectures and problems that may arise during modelling, two versions of the agents were implemented; A Soar agent and an Agent Factory agent. Both agents were conducting the same type of mission and was built on the same set of theories. The Soar version was written with Soar 8.2 and Tcl/Tk 8.0 and Visual Soar. The Agent Factory version was written with Agent Factory, Visual OMT and Java.

### 4.1 Soar agent

As mentioned in chapter 2 the agent should be able to act in an environment with other participants. To achieve this, a small simulation system was created. The system consisted of a central server and several clients, called platforms. The platforms ran their own simulation systems, simulating different military platforms such as radar units, tanks and aircrafts. The sole function of the server was to handle communication between different platforms.

#### 4.1.1 Platforms

The platforms connected to the server by registering the name of the platform, the number of units, type of platform and radar-range. The number of units was added so that a large number of platforms could be treated as one client. After registering, all communication was initiated by the server, using a tick-based control. In every tick, the server sent a tick-command to let the platforms advance their simulations one tick (see appendix). When the platforms had made this advance they sent their position back to the server as well as a done-message. Based on radar ranges and positions the server then notified the clients of other visible platforms, called contacts, by sending the position, name and type of the contacts. Since each platform

ran their own simulation, each platform had their own graphical interface to visualise their interpretation of the environment.

Sockets handled all communication between the server and the platforms. To reduce network traffic and processing, position updates were only sent from the platforms to the server when the platforms had changed their positions since the last update. Also, the server sent information about visible contacts only to those platforms that did not already have this information. This required more processing at the server since it had to make sure that the platforms had consistent information. The server had to tell the platforms to add, remove or update contact data appropriately. Since some platforms, such as radar stations, rarely moved, they only needed to send their position once to the server. In addition, moving platforms that discovered a non-moving contact, only received one update when the contact came within radar range and one message to remove the contact when it no longer was within radar range.

The client-server approach was chosen so that it would be easy to add and remove platforms at any time during the simulation, making it easier to evaluate the behaviour of the agent in different situations.

## Radar

Radar platforms were objects that did not move, they only waited for other platforms to appear. As soon as another contact platform appeared within the radar range, the radar platform showed the name and position of that contact. No other processing occurred at a radar platform.



The image shows a window titled "RadarGroup1" with a standard Windows-style title bar. Below the title bar, the text "Position: 35000 65000" is displayed. Underneath, the word "Contacts" is centered. A table with two columns, "Name" and "Position", is shown. The table contains one row with the name "ac-2" and the position "38271.0564432,64256.6510158".

Name	Position
ac-2	38271.0564432,64256.6510158

Figure 4.1: The graphical interface for the radar platform.

## Aircraft

Aircraft platforms were simulations that continuously calculated flight parameters from a simple flight model and sent the resulting position to the server. A position was expressed in 2-dimensional RT90<sup>1</sup> coordinates and an additional height component. A new position was calculated from the previous position, heading, speed and angle of attack. Speed was calculated from current speed and thrust. Angle of attack and heading were calculated from

---

<sup>1</sup>Rikets koordinater

previous values and the position of the joystick and thrust was calculated from the position of the thrust control.

When the simulation started, it loaded flight instructions such as maps, way-points, target type and location, mission type and initial flight parameters. The instructions were loaded from a script file to easily create new scenarios. A graphical interface visualised the simulation by drawing a moving map with its centre on the position of the aircraft, similar to the moving maps found in a cockpit. The map showed land and water areas, contacts, current heading, position of the aircraft and position of way-points as well as the flight path between the way-points (see figure 4.2). Note that different aircrafts loaded their own missions and their own maps.

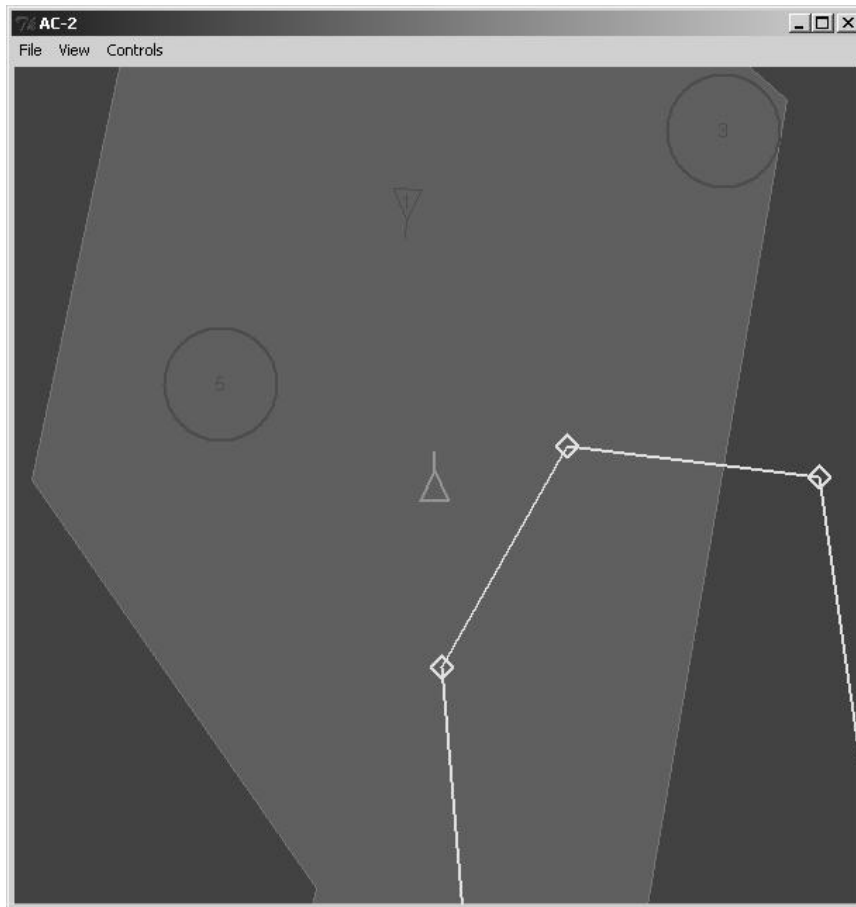


Figure 4.2: A graphical interface for the aircraft platform visualizing terrain, waypoints and radar contacts.

In addition to update positions, the aircraft also had a computer to continuously calculate data on the current mission. The computer held informa-

tion about the type of mission, position and name of the target, way-point information such as number, position and commanded altitude. The computer also continuously calculated distance and heading to the next way-point and a heading bug that displayed the deviation between current heading and heading to the next way-point. The heading deviation was quantified into seven levels and is further discussed in 4.1.2. The computer automatically switched to the next way-point when the distance to current way-point was less than some predefined value, as on most military aircraft computers. If a contact was reported from the server, the aircraft computer also calculated the distance, heading and heading deviation to the contact. The aircraft model also provided an interface to external programs or modules. All the data calculated by the flight model and the aircraft computer was provided through a cockpit interface. The cockpit interface consisted of four different displays or monitors. To receive data from the aircraft the external program simply called one of those four display functions. The interface also provided two functions for moving the joystick and the thrust control. The interface was thus similar to a real cockpit interface and provided a clear separation between the aircraft and the external module i.e. agent.

#### 4.1.2 Agent

Agents could only be assigned to aircraft platforms and were started by creating child interpreters in Tcl. All communication between the aircraft and the agent was done by interprocess communication. The role of the agent was to control the aircraft by using the aircraft interface (see 4.1.1). The agent perceived the simulation environment by calling the display functions and it controlled the aircraft by calling the joystick and the thrust control functions. All reasoning was done through the embedded Soar engine and its production rules.

In every cycle the agent called one of the display functions. The data received was put on the input-link of the agent. Since the input-link was part of the working memory of the agent, changes could cause some production rules to fire. When all rules had fired, the agent might put some actions on its output-link. Those actions were then sent back to the aircraft as either a joystick command, a thrust control command or both.

Going from a natural phenomenon in the real world to computational models, a lot of simplifications have to be made, especially mapping continuous attributes and events to discrete values. In this case one such mapping needed to be done for the heading bug. The heading bug is a small rectangle on the HUD screen that shows the relative direction to the next way-point, target, etc. If the bug is horizontally centred on the HUD the current heading will lead to the way-point. If the heading is off-course the heading bug is positioned to the right or to the left of the HUD. The larger the course deviation, the further the bug moves from the centre of the HUD. How should

Decoded region	Region in words
-3	Much to the left
-2	To the left
-1	A little to the left
0	About or exactly centre
1	A little to the right
2	To the right
3	Much to the right

Table 4.1: A discrete representation of the position of the heading bug.

the position of the heading bug be decoded into discrete symbols. One could count the number of pixels on the screen between the bug and the centre. This is not very realistic and not adopted by any pilot. The theory chosen here states that humans tend to create discrete abstractions naturally. The bug is not so or so many millimetres to the right, it is a little left or much left on the screen. Humans also tend to use relative measures such as, it is more left than previously. With this in mind the heading bug was divided into seven regions, here decoded into integers (see table 4.1). The same theory and technique was used for controlling the joystick and the thrust controller.

One way to look at Soar is as an architecture for cognitive modelling. However, Soar lacks some aspects that might be needed for modelling human behaviour. For these reasons Soar has been used in different hybrid architectures, where the Soar engine has been used to model decision making and another architecture has been used to model detailed perception and/or motor commands. Since the agents in this study had some special requirements that would be difficult to model in Soar, it was decided to build an independent psychological as well as a perceptual engine. The engines were built as separate modules but were able to exchange data and thereby affect each other. For instance, the psychological engine could model stress which affected the visual scan pattern modelled in the perceptual engine. Both engines worked in conjunction as a filter to modify perceptual information and motor commands from/to the environment.

### Perceptual engine

The perceptual engine was built from the theories presented in 3.3.4. It chose which cockpit display the agent was looking at by using Markov models and then called the corresponding display function from the aircraft interface. By having the engine to work outside the decision system, is just like saying that the agent can not control its shift of attention by decision. This does not sound very reasonable, so the agent had an option to choose between different scan strategies. Each strategy had its own Markov model. For instance, a

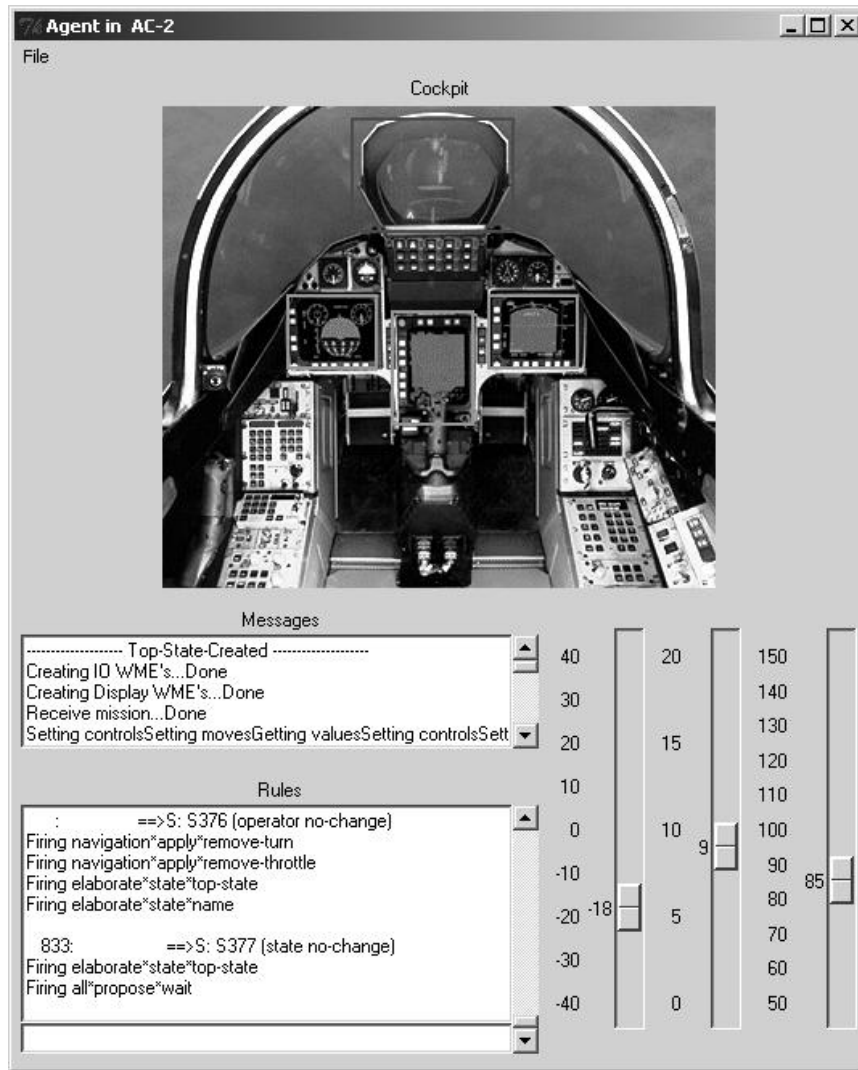


Figure 4.3: The graphical interface for the agent. The interface shows which display the agent is currently looking at, heart rate, number of perceived targets and its perceived altitude. In the lower text box the decisions are written.

scan strategy called “normal operation” had a Markov model where the agent spent most of the time looking at the HUD and sometimes glanced at the other displays. This model was used when the agent did normal route flying. Another strategy, “hud”, was used within very short range of the target, in which the agent spent all his time looking at the HUD. It was also possible for the psychological engine to modify the random distributions of the Markov model. For instance, when the agent chose “normal operation” scanning, the psychological engine could affect the scan pattern so that the agent spent less time looking at the HUD display, typically when many contacts were present.

### **Psychological engine**

The psychological engine was supposed to run a computational Lisrel model and thereby implement a psychological model founded on experimental data. Unfortunately the complete Lisrel models were not available at the time of this study. Another idea was to use neural nets, which would produce a similar system as that of the Lisrel models. As a substitute, simple relationships were used. The number of contacts were reduced so that every fifth target was not relayed to the agent. Heart rate was modelled as a function of distance (or time) to the ground target and number of threats on the radar. The function was created from analysis of heart rate vs time curves from studies on attack missions. The perceived altitude was also modified so that the agent climbed two meters per additional contact after seven. Since the goal was to use Lisrel models with parameters such as workload, situational awareness, etc, some additional questions arose. Should the agent be able to reason about these factors? Does an agent know the status of his workload, awareness, stress, etc? If so, does this knowledge reflect its behaviour? Since, the psychological engine worked as a filter and had access to the working memory, it could have added these factors for the agent to reason about. However, from the studies no behaviour seemed to use this information so it was decided that these factors could not be reasoned about. Should the agent be able to affect its psychological status? Since it was not capable to reason about psychological factors it could not affect them either, a theory stating that humans inherit some internal phenomena that they can not control by decisions. This might sound reasonable in some situations and less likely in others. Sometimes a human takes measures to calm down or make decisions to put them in less stressful situations. Sometimes an individual does not know that it is mentally challenged. The psychological engine did not only work as a filter for the perceived objects. It was also designed to modify actions, simulating physical limitations such as precision.

## Perception and actions

In every tick, the Soar input function called the perceptual system to extract objects from the environment. When the perceptual system and the psychological system had applied their filters the modified set of objects were translated into working memory elements in Soar. Since the Soar production rules fire because of changes in the working memory, continuous updates of existing values would cause excessive firings. To avoid this, the input function held a table of attribute-value pairs that matched the current working memory of the agent. The input function checked every value it received from the aircraft to inspect whether the agent needed to be updated. Sometimes, the aircraft lost information, e.g. a contact was lost. When this happened the lost attribute was assigned a value of "nil". When the agent received a "nil" value the corresponding WME was removed. There were five possible outcomes:

1. Attribute and value is not known by the agent -> Add new attribute and value
2. Attribute is known but value does not match -> Remove old attribute and add new
3. Attribute is known but should be removed -> Remove old attribute
4. Attribute is not known and should be removed -> Do nothing
5. Attribute and value is known by the agent -> Do nothing

When the input function had updated the working memory, the Soar engine executed the production rules. In Soar the reasoning is goal-driven; the agent always has a goal to achieve. A goal can also be divided into several sub-goals whose purpose is to achieve higher goals. Part of the goal-hierarchy diagram can be found in A.1.

As all reasoning was done, the agent might have chosen to act. The only actions that the agent was capable to perform were to control the joystick and the thrust control. These actions were put on the output-link. The output-function then sent these commands by calling the corresponding functions from the aircraft interface. To enable the agent to work asynchronously with the aircraft, the output function also added a "status complete" flag to tell the agent that the command had been sent. When an action received a "status complete" flag, the agent then removed the command.

### 4.1.3 Future enhancements

The motivation is of course not to create a complete artificial human, but a model can always be made to perform slightly better; in this case, much

better. Some implementation ideas that emerged during the implementation were:

**Prediction** How should the agent be able to foresee events? How should dead reckoning or planning be modelled? Some implementations have used the internal knowledge of the agent to achieve this. The agent has mentally stepped inside the opponents position and applied its own knowledge to foresee the opponents actions.

**Visual engine** Here, the agent could only perceive the environment through displays. What if he looks out of the windshield? What if there is an agent running around in the wood, with lots of trees and bushes? Clearly, a very sophisticated visual engine must be created.

**Psychological engine** As mentioned, Lisrel models or neural nets can be used to create a model for decreased (or increased) capabilities. They are also suitable for using experimental data from studies on human counterparts.

**Simulation systems** Because many military simulations use HLA to communicate, it is preferable to have an agent using the same standard. It is therefore interesting to create such an interface on top of the Soar engine.

## 4.2 Agent Factory agent

The Agent Factory agent had basically the same behaviour as the Soar agent, but lacked some features such as choosing scan patterns, fly a patrol mission. An existing flight model was used to simulate the aircraft. A graphical 2-D map (see figure 4.4) as well as a goal-plan chart (see figure 4.5) was also used to monitor behaviour and were provided by Pitch. These components and the agent were built on an HLA interface so all communication was provided by the RTI.

### 4.2.1 Platforms

As in the Soar simulation system, all platforms ran their own simulations. All the platforms<sup>2</sup> were HLA compatible and used RTI to communicate. A single simulation that can interact with other simulations is called a federate in an HLA environment. Every federate may publish objects or attributes that other federates might be interested in. Also, every federate can subscribe to published objects or attributes. This architecture makes it possible to interact with several simulations in a distributed manner. No radar platforms was implemented in the Agent Factory system, only aircraft platforms.

---

<sup>2</sup>Federates in HLA terms



Figure 4.4: A graphical 2D map. A map application, being a federate, listening to the RTI for aircraft positions and plotting the aircrafts on a map.

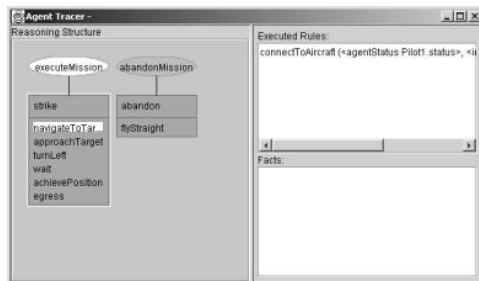


Figure 4.5: A GUI displaying current goals, plans and executed rules. This interface gives an opportunity to track the agent's thoughts and actions.

## Aircrafts

The aircraft federate ran a simulation that calculated flight parameters just like the platform in the Soar example. To let other federates interact with the aircraft it subscribed to a joystick and a thrust object. Other federates could thus publish a joystick object with stick motion values to control the aircraft. As in the Soar example the aircraft federate published data from its computer. It was thus possible for other federates to subscribe to this information.

### 4.2.2 Agent

As Agent Factory provides a graphical interface, it required much less effort to express behaviour. All objects that the agent was able to reason about had to be defined in advance. This was done with an object modelling tool Visual OMT<sup>3</sup>. Perception was done by simply subscribing to data published by the aircraft. This was done by a sensor module. The sensor module then filtered out information that the agent was not currently looking at. No psychological engine was built, but a perceptual engine was included in the sensor system to model shift of attention. The engine was written in Java, and used Markov models to randomly shift attention. The agent was not capable to choose between scan patterns as in the Soar implementation. This was due to time constraints as well as some architectural limitations<sup>4</sup>.

---

<sup>3</sup>Provided by Pitch Kunskapsutveckling AB

<sup>4</sup>Pitch is currently working on a new version of Agent Factory where the architecture has changed to overcome these limits.

## Chapter 5

# Conclusions

A CGF entity can be created in lots of different ways, depending on the needs and resources available. Since the study is part of a project with a vision to create a library of entities acting in different domains, it is highly influenced by a search for general and uniform modelling techniques. These techniques or methods are rooted in a traditional view of cognition, because that tradition favours such methods. The study presents a theory of mixing rationality and cognitive constraints to create more realistic behaviour. It is thus compatible with validation, verification and task analysis, techniques that are used more often in the context of modelling and simulation.

During the development and research of CGF's, a lot of cognitive theories unfold. Which ones are to be used? Some theories may appear better than others in different domains and for different fidelities. What is suggested here is to have an open mind. The CGF that was created in this study was made up of components: a visual system, a psychological system and a decision system. In such a system, components can be replaced, added, removed and maybe reused in different models. The study have identified three central areas in CGF development, where cognition is a part of all three.

The prototypes modelled here, were not evaluated by any subject matter experts. From a non-expert evaluation the random scan pattern seemed to match possible scan pattern outcomes. The agents also had to do small corrections to the flight course, instead of turning to the exact course in one try. Sometimes the agents turned a little too late because they were attending "wrong" instruments. In critical situations when the agents where focusing on the course, no such errors were made.

**Task analysis** To create realistic behaviour, the tasks and the role of the agent must be investigated. The HTA that was used here, worked as a template for acquiring data and also a method of documenting that data. The documentation provided a layer between data acquisition and implementation. HTA is certainly a recommended method to use in future modelling,

although the decomposition categories need to be tuned for the current tasks. An idea is to further divide the decomposition categories into a general and a task dependent group.

**Modelling tools** The choice of architecture is highly dependent on the current task. The Soar architecture provides a scalable and dynamic representation of behaviour. These features makes it a tool for modelling tasks with a full range of complexity. Agent Factory has a less dynamic representation of behaviour but has a preferable interaction system that simplifies communication with other entities and environments. Agent Factory provides a nice modelling environment with a graphical user interface, while Soar is more difficult to learn. For less complex tasks the recommended architecture is Agent Factory. For more dynamic and complex tasks Soar is the preferred choice, although a lot of effort has to be made when creating the interaction system.

Since not all architectures have sophisticated mechanisms for visual, auditory and motor systems, some extensions might need to be created. The extensions and simplifications in this study are examples of how to achieve more complete human behaviour. Such extensions will certainly be useful in other domains as well.

The Soar prototype created in this study was extended with a psychological model as well as visual model. The suggestion is to use the Soar engine as it is used here; a medium for reasoning, decision-making and planning. The Soar programming language alone, does not provide enough flexibility to model complex models of vision, fear, etc. As mentioned earlier, a solution would be to connect extension components to the decision engine and provide data for the decision engine. For example, the psychological component could produce some value of fear. If the modeller wants the agent to be able to reason about its own level of fear the value is sent to the decision engine. The fear level could also be sent to a motor or action component. That component could then reduce the precision by modifying actions. By letting the components interact with each other, some psychological parameters may never reach the decision making component but may still affect other components and behaviour.

In Agent Factory the Java language may be used to create more complex computational models. It is thus possible to implement such functionality within the rule expressions. From a Software engineering perspective a component based approach, as introduced earlier, is a more suitable approach. Components could easily be changed and managed to try different theories and parameters. Since Agent Factory agents are HLA-compatible, all components could be created as federates exchanging information. The next version of Agent Factory is supposed to provide an architecture where all components are modelled as federates and are part of an internal federation.

**Psychological models** The psychological models used here are not sufficient to create more realistic behaviour. Extending the models with additional parameters describing performance modifiers and physical actions is necessary. The models used here have been developed after numerous experiments and studies on military aircraft pilots. In different domains and tasks, such models might not be available. If models do not exist, there is also a need to find methods to develop similar models to those presented here. Since many methods of finding such models use statistical analyses, the CGF's will inherit statistical relationships, not individual characteristics. As a modeller, one must be aware of the consequences that may arise and find methods to avoid stereotype and predictable behaviour.

# Appendix A

## Soar agent

### A.1 Goal hierarchy

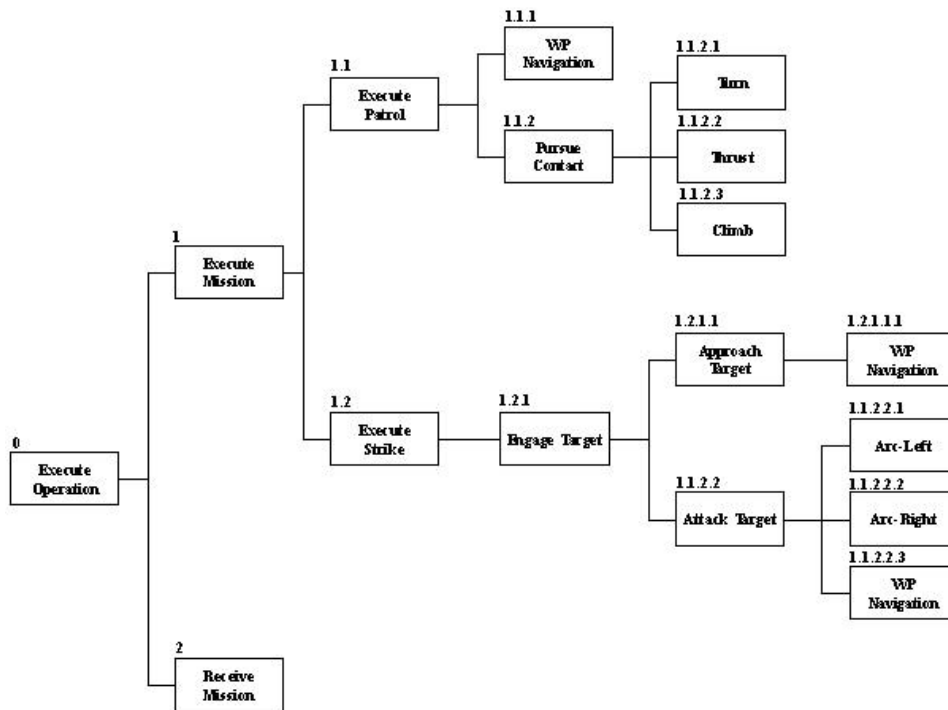


Figure A.1: The goal hierarchy for the agent

**A.2 Input-link**

**A.3 System**

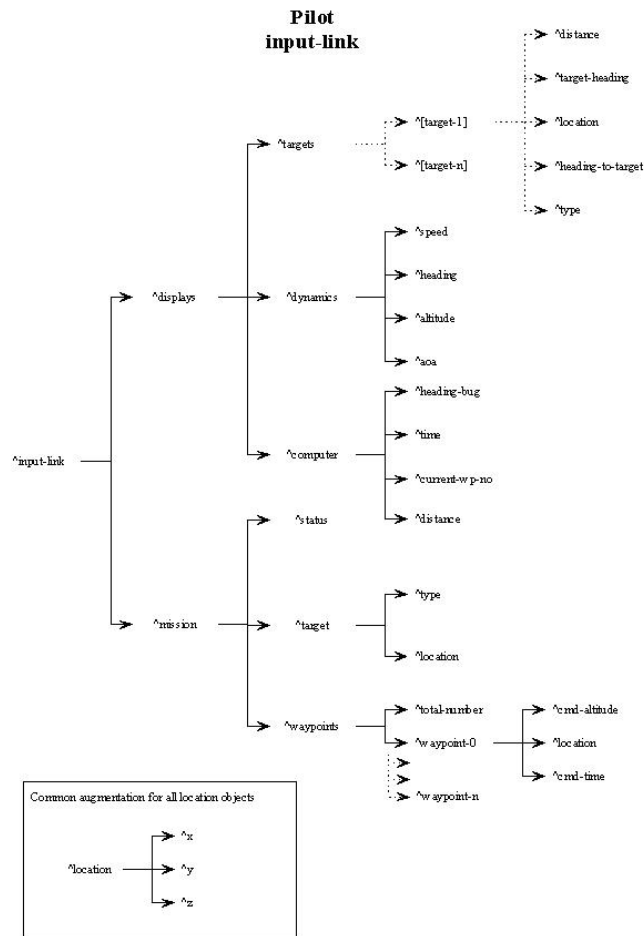


Figure A.2: The perceived information from the environment.

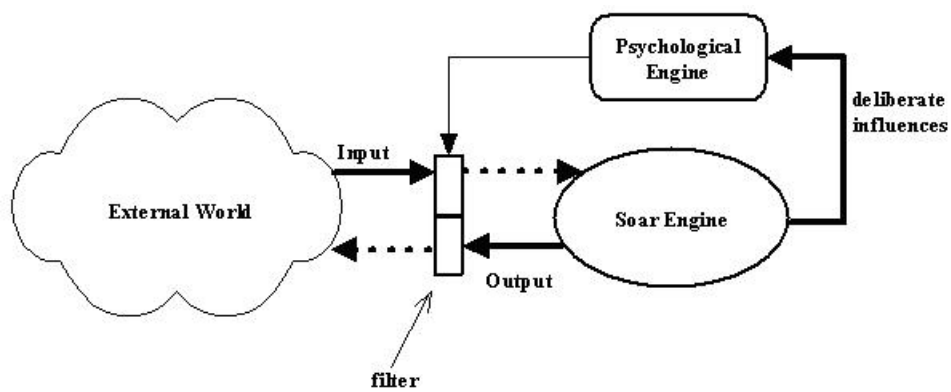


Figure A.3: The agent engine

# Bibliography

- [AJ] Jennie Andersson and Magnus Johansson. Agent factory - concept and development environment. Technical report, Pitch Kunskapsutveckling.
- [AJL] Jennie Andersson, Magnus Johansson, and Staffan Lof. Modeling component-based intelligent agents in an hla-environment using an agent development tool. Technical report, Pitch Kunskapsutveckling.
- [AL] Jennie Andersson and Staffan Lof. Hla as conceptual basis for a multi-agent environment. Technical report, Pitch Kunskapsutveckling.
- [Ber00] Peter Berggren. Situational awareness, mental workload, and pilot performance - relationships and conceptual aspects. Technical report, Human Sciences Division, FOA, 2000.
- [Cho01] Ronald S. Chong. Low-level behavioral modeling and the hla: An epic-soar model of an enroute air-traffic control task. Technical report, Soar Technology Inc, 2001.
- [CS95] Clare Bates Congdon and Karl B. Schwamb. The soar advanced applications manual. Draft 7, Electrical Engineering and Computer Science Dept., University of Michigan and Information Sciences Institute, University of California, 1995.
- [DBDH<sup>+</sup>01] Melaine Diez, Deborah A. Boehm-Davis, Robert W. Holt, Mary E. Pinney, Jeffrey T. Hansberger, and Wolfgang Schoppek. Tracking pilot interactions with flight management systems through eye movements. Technical report, George Mason University, 2001.
- [Doma] Uwe Dompke. Computer generated forces - background, definition and basic technologies. In *Simulation of and for Military Decision Making*.

- [Domb] Uwe Dompke. Human behaviour representation - definition. In *Simulation of and for Military Decision Making*.
- [DS00] Stephanie M. Doane and Young Woo Sohn. Adapt: A predictive cognitive model of user visual attention and action planning. In *User-Modelling and User-Adapted Interaction (UMUAI)*, volume 10. Kluwer Academic Publishers, 2000.
- [Har] Randall L. Harris. Effects of foveal information processing. Technical report, NASA Langley Research Center.
- [Hil] Randall W. Hill. Modeling perceptual attention in virtual humans. Technical report, Information Sciences Institute.
- [Hum98] *Modeling Human and Organizational Behavior: Application to Military Simulations*, chapter Attention and Multitasking. The National Academy Press, 1998.
- [JERTa] Randolph M. Jones, John E.Laird, Paul S. Roosenbloom, and Milind Tambe. Generating behavior in response to interacting goals. Technical report, Artificial Intelligence Laboratory, University of Michigan and Information Sciences Institute, University of California.
- [JERTb] Randolph M. Jones, John E.Laird, Paul S. Roosenbloom, and Milind Tambe. Intelligent automated agents for flight training simulators. Technical report, Artificial Intelligence Laboratory, University of Michigan and Department of Computer Science & Information Sciences Institute, University of California and School of Computer Science, Carnegie Mellon University.
- [JLN+99] Randolph M. Jones, John E. Laird, Paul E. Nielsen, et al. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 1999.
- [JWvLL] Randolph M. Jones, Robert Wray, Michael van Lent, and John E. Laird. Planning in the tactical air domain. Technical report, Artificial Intelligence Laboratory, University of Michigan.
- [KA89] B. Kirwan and L.K. Ainsworth, editors. *A guide to task analysis*. Taylor & Francis, 1989.
- [Kle99] W. Bastiaan Kleijn. Hidden markov models for speech recognition. Technical report, Speech, Music and Hearing, KTH, 1999.

- [Lai01] John E. Laird. The soar 8 tutorial. Technical report, University of Michigan, February 2001.
- [LCC99] John E. Laird, Clare Bates Congdon, and Karen J. Coulter. The soar user's manual. Technical Report 8.2, Electrical Engineering and Computer Science Dept., University of Michigan, 1999.
- [Lew99] Richard L. Lewis. Cognitive theory, soar. Technical report, Dept. Computer & Information Science, 1999.
- [LLR] Jill Fain Lehman, John Laird, and Paul Roosenbloom. A gentle introduction to soar, an architecture for human cognition. Technical report.
- [Mil56] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.
- [Nev97] Robert Nevalainen. A cognitive model describing the behaviour of a fighter pilot. Master's thesis, Linkopings Universitet, 1997.
- [New90] Allen Newell. *Unified theories of cognition*. Harvard University Press, 1990.
- [NLE98] Cornelia Niessen, Sandro Leuchter, and Klaus Eyferth, editors. *A Psychological Model of Air Traffic Control and Its Implementation*, 1998.
- [RA00] Frank E. Ritter and Marios N. Avraamides. Steps towards including behavior moderators in human performance models in synthetic environments. Technical report, School of Information Sciences and Technology, The Pennsylvania State University, 2000.
- [Rit01] Frank E. Ritter. Soar. *Encyclopedia of Cognitive Science*, 2001.
- [RJ86] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 1986.
- [SAT+92] Erland Svensson, Maud Angelborg-Thanderz, et al. Information overflow - mental workload and performance in military aviation. Technical report, Department of Human Studies, Swedish Defence Research Agency, 1992.
- [SBL00] Dario D. Salvucci, Erwin R. Boer, and Andrew Liu. Toward an integrated model of driver behavior in a cognitive rchitecture. Technical report, Four Cambridge Center and Mass. Institute of Technology, 1990.

- [SF99] Erland A.I. Svensson and Glenn F. Wilsson. Psychological and psychophysiological models of pilot performance for system development and mission evaluation. Technical report, Swedish Defence Research Agency and US Air Force Research Laboratory, 1999.
- [Tac97] *Tactical Display for Soldiers: Human Factors Considerations*, chapter Stress and Cognitive Workload. The National Academy of Science, 1997.
- [Tol] Andreas Tolk. Computer generated forces - integration into the operational environment. In *Simulation of and for Military Decision Making*.
- [win86] *Understanding Computers and Cognition*. Ablex Corporation, 1986.
- [YL] Richard M. Young and Richard L. Lewis. The soar cognitive architecture and human working memory. Technical report, Dept. Psychology, University of Herfordshire, Hatfield, Dept. Computer & Information Science, Ohio State University, Ohio.