
	<div>HUMAN</div> <div>Model-based Analysis of Human Errors during Aircraft Cockpit System Design</div>	
Virtual Simulation Platform Description		
<div>Full Title:</div> <div>Document ID:</div> <div>Nature:</div> <div>Classification:</div> <div>Version:</div> <div>Parts & Classifications:</div> <div>Work Package:</div> <div>Milestone:</div> <div>Document Timescale:</div> <div>Start of Document:</div> <div>Final version due to:</div> <div>Time now:</div> <div>Scope:</div>	<div>Description of the cognitive crew model and virtual simulation platform</div> <div>HUMAN/WP3/OFF/D3.4</div> <div>Deliverable</div> <div>Public with Confidential Annex</div> <div>V1.1<div>Saved: 06/07/09</div></div> <div>Main document (Public)</div> <div>Annex:<div>I. Acceptance Test Results VSP (Confidential)</div></div> <div>WP3</div> <div>M2</div> <div>Project Start Date: March 1, 2008</div> <div>T0+07</div> <div>T0+14</div> <div>T0+16</div> <div>To discuss and describe the cognitive architecture and the virtual simulation platform</div>	
<div>Keywords:</div>	<div>Cognitive Architecture, Virtual Simulation Platform</div>	
<div>Issue Date (dd/mm/yyyy):</div>	<div>06/07/2009</div>	
<div>Compiled:</div>	<div>Jan-Patrick Osterloh, OFF</div>	
<div>Authors:</div>	<div>OFF: Jan-Patrick Osterloh, Tomasz Mistrzyk, Andreas Lüdtkke</div>	
	<div>TNO: Tina Mioch, Rosemarijn Looije,</div>	
	<div>BCH: Juan Manuel González Calleros</div>	
<div>Technical Approval:</div>	<div>Jan-Patrick Osterloh, OFF</div>	
<div>Issue Authorisation:</div>	<div>Andreas Lüdtkke, OFF</div>	
<div>© All rights reserved.</div> <div>This document is supplied by the specific HUMAN work package (Proj. No: 211988) quoted above on the express condition that it is treated as confidential to those specifically mentioned on the distribution list. No use may be made thereof other than expressly authorised by that work package leader.</div>		

DISTRIBUTION LIST		
Copy type ¹	Company and Location	Recipient
C, E	European Commission DG RTD - Directorate H - Transport CDMA 4/138 B-1049 Bruxelles	Stephanie Stoltz-Douchet
T	All HUMAN partners	All HUMAN partners

¹ Copy types:
 M = Master copy,
 E = Email,
 C = Controlled copy (paper),
 D = Electronic copy on disk,
 T = TeamSite (Sharepoint)

RECORD OF REVISION			
Version	Date (dd/mm/yyyy)	Status Description	Author
V0.1	30/09/2008	Initial Version	JPO
V0.2	03/10/2008	Comments by ALA	AC
V0.3	10/11/2008	Added content for the scenario controller	TiM
V0.4	03/12/2008	Added content for Cognitive Layer	TiM
V0.5	09/12/2008	Added content for Cognitive Architecture	JPO
V0.6	09/02/2009	Communication Model	TM
V0.7	09/03/2009	Task Modelling - state of the art	TM
V0.8	12/03/2009	AMBOSS+PED	TM
V0.9	12/05/2009	Added Percept and Motor description, spell check	JPO
V0.91	13/05/2009	Added SHAMI description	JMGC
V0.92	13/05/2009	Extending content of cognitive layer	TiM
V0.93	03/06/2009	Attentional capture changed	RL
V0.94	18/06/2009	Integrated Comments	TiM
V0.95	18/06/2009	References formatted, integration of comments	JPO
V1.0	03/07/2009	Final Review	JPO
V1.1	06/07/2009	Final check	AL

[illegible]

1 Table of Contents

1	Table of Contents.....	5
2	Abbreviations and Definitions.....	7
3	Introduction	8
3.1	Architecture of the VSP	8
3.2	Requirements for the VSP	10
4	Task Analysis and Task Modelling.....	11
4.1	Requirements from WP1 for PED	12
4.2	Theoretical foundations for HUMAN Task Modelling	13
4.3	HUMAN Task Modelling approach.....	14
4.4	Model of HUMAN Task Modelling tool chain	16
4.4.1	Modelling environment for Semi-Formal Task Models	16
4.4.2	Modelling environment for Formal Task Models.....	19
4.4.3	Rules based language	20
4.4.4	Implementation Status of the HUMAN Formal Task Modelling	22
5	Cognitive Architecture	24
5.1	Theoretical Foundation for the Cognitive Architecture	25
5.2	Model of the Cognitive Architecture	26
5.2.1	Cognitive Architecture	27
5.2.1.1	Crew Coordination.....	27
5.2.1.2	Visual Perception	32
5.2.1.3	Motor	37
5.2.1.4	Memory	41
5.2.2	Associative Layer.....	50
5.2.2.1	Reactive Behaviour.....	50
5.2.2.2	Multitasking.....	51
5.2.1	Cognitive Layer	55
5.2.1.1	Decision Making Module	56
5.2.1.2	Task Execution Module	59
5.2.1.3	Sign-Symbol Translator	60
5.2.1.4	Monitoring the Associative Layer	62
5.2.1.5	Cognitive lookup	63
5.2.2	Inter-layer communication.....	65
5.2.2.1	Example of the inter-layer communication.....	65
5.3	Status of the Cognitive Architecture	68
6	Virtual Simulation Platform	72
6.1	Scenario Controller.....	72
6.1.1	Requirements for the Scenario Controller.....	72
6.1.2	Model of the Scenario Controller.....	74
6.1.3	Scenario Script.....	74
6.1.4	Traffic and Weather Module	75

6.1.5	Scenario Interpreter.....	75
6.1.6	Interface to the Data pool.....	76
6.1.7	Status of the Scenario Controller	77
6.2	Symbolic AHMI	77
6.2.1	Requirements.....	78
6.2.2	Implementation status	80
7	Summary	81
8	References.....	83

Annex I: Acceptance Test Results VSP

2 Abbreviations and Definitions

Abbreviation	Definition
A/C	Aircraft
AFMS	A dvanced F light M anagement S ystem
AHMI	A dvanced H uman M achine I nterface
AL	Associative Layer
AOI	A rea o f I nterest
ATC	A ir T raffic C ontroller
CASCaS	C ognitive A rchitecture for S afety C ritical T ask S imulation
CL	Cognitive Layer
CLIPS	Rule Based Language used in Cognitive Layer and Scenario Contoller
Cognitive Architecture	Simulation software that can interpret normative behaviour in order to predict/model human behaviour
Cognitive Crew	Crew build by two instances of CASCaS, with procedures for PF and PM
Cognitive Model	Combination of a Cognitive Architecture (CASCaS and a procedure model)
CPDL(C)	Controller Pilot Data Link (Communications) - method by which air traffic controllers can communicate with pilots over a datalink system
CTT	C oncur T ask T ree
ENG	E ngine Display
EPM	E rror P roduction M echanism
ET	E rror T ype
FL	F light L evel
FMA	F light M ode A nnunciation
GOMS	G oals O perators M ethods and S election rules
HEA	H uman E rror A nalysis
HTA	H ierarchical T ask A nalysis
LTM	L ong T erm M emory
ND	N avigation D isplay
PF	P ilot F lying
PFD	P rimary F light D isplay
PM / PNF	P ilot M onitoring / P ilot n on F lying
PSP	P hysical S imulation P latform
SST	S ign to S ymbol T ranslator
TSC	T ime S witch C ost
VSP	V irtual S imulation P latform
WM	W orking M emory

3 Introduction

Objective of this document is to describe the Virtual Simulation Platform (VSP). In HUMAN, the VSP will be used to predict crew behaviour, including errors, for the scenarios defined in WP1 and WP2. In WP4, this behaviour will be compared with real crew behaviour, deduced for the same scenarios. Based on the comparison between predicted and actual behaviour, new requirements for the VSP will be deferred. This document describes the architecture of the VSP and the cognitive model. In the next section, the general architecture of the VSP and its modules are described.

3.1 Architecture of the VSP

Figure 3-1 shows the architecture of the VSP in HUMAN:

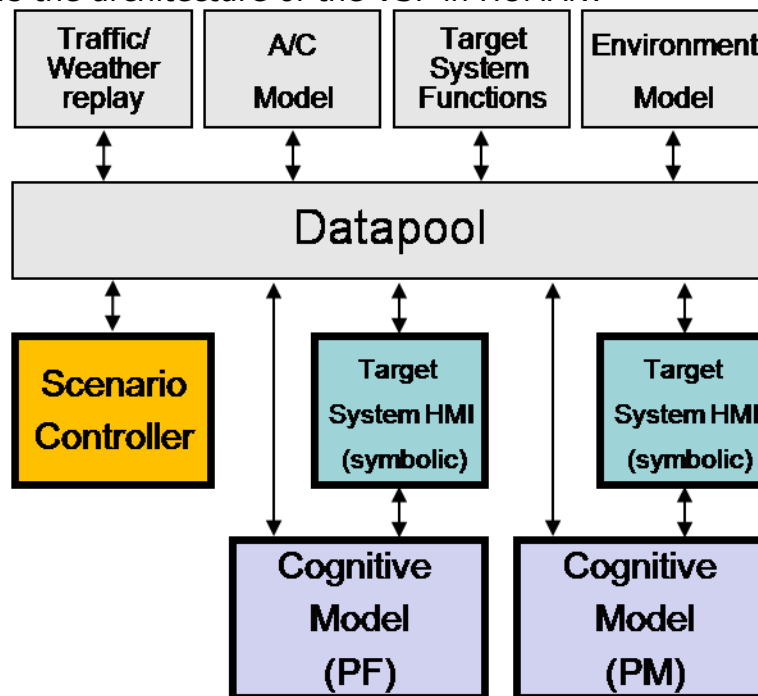


Figure 3-1: HUMAN Simulation Platforms Architecture

In the following, the modules of the VSP as depicted in Figure 3-1 should be described. All grey modules of Figure 3-1 are of general applicability, and a detailed description of them will be in D2.4 and D2.8. The other modules, depicted in colours and a bold box, are HUMAN specific and will be described in this document in more detail.

The *traffic/weather* module will replay pre-recorded traffic and weather (thunderstorms), according to the scenario (see scenario controller). The recoding of the traffic and the weather will be done by DLR, as part of the scenario design. The module will be maintained by DLR.

The scenarios which are flown by the cognitive models (and the human pilots on the PSP) are controlled by the *scenario controller*. This includes mainly

- starting the simulation software (A/C model, environment, datapool, traffic/weather replay, ...),
- starting the cognitive models (one for PF and one for PM),
- determining the end of scenario and start the cogn. models learning phase,
- stopping the simulation software and the cognitive models,
- restarting everything in batch mode.

For further discussion of the Scenario Controller, see section 6.1. This module is developed by TNO in cooperation with DLR and OFF.

The *A/C module* simulates the aircraft behaviour (e.g. speed, pitch, roll), according to the physical performance of the aircraft and its actual parameters (e.g. thrust, flaps, slaps) with respect to external conditions, like actual weather conditions (e.g. wind), or the terrain, which are controlled by the *environment model*. Both modules have been developed and are maintained by DLR. The *target system* (main system that will be investigated in HUMAN) is the AHMI (Advanced Human Machine Interface), the User Interface for a 4D Flight Management System, called AFMS (Advanced Flight Management System). The AFMS functionality is represented as "Target System Function" module in Figure 3-1. For the VSP, the AHMI will be replaced by a *symbolic AHMI*, which allows the Cognitive Model to interact with the AFMS. This is needed, as the Cognitive Model has no real eyes and hands, and thus cannot interact with the real graphical user interface itself. The AHMI and the AFMS have been developed by DLR, and the symbolic AHMI will be developed by BCH.

All modules of the simulations platforms (VSP and PSP) communicate with each other via a dedicated module, the *datapool*. The datapool not only passes messages between the modules, but is also the "flight recorder" (Black Box), which records all information that is necessary to replay the flight.

Last but not least, the VSP consists of two Cognitive Models, one for the pilot flying (PF) and one for the pilot monitoring (PM). In HUMAN, a cognitive model is understood as the combination of the pilot's knowledge (the normative behaviour plus additional knowledge needed to fly an A/C) and the cognitive architecture that interprets this knowledge in order to achieve a certain goal, e.g. flying the A/C in a scenario. In order to derive the normative behaviour, a task analysis is needed. This step will be described in section 4. The cognitive architecture is described in more detail in section 5. The scenario controller will be described in section 6.1, and the symbolic AHMI in section 6.2.

3.2 Requirements for the VSP

In WP1, several requirements for the VSP and basic capabilities for the cognitive architecture have been derived (see Deliverable D1.3), and in D1.2 the Error Production Mechanisms of the cognitive architecture have been specified. The following table summarises these requirements for the cognitive architecture:

Req. No.	Description
CM.01	The VSP must be able to simulate a crew, consisting of PF and PM. Therefore the platform must support communication between PF and PM, restricted to simulated verbal communication.
CM.02	The VSP must be able to interpret crew (PF, PNF) normative procedures.
CM.03	The perception of the cognitive architecture (CA) must be sophisticated, in terms of a restricted visual field with focus, as well as eye-movements. Perception must support the detection of certain modelled events, e.g. warning lights or sounds, which trigger reactive rules.
CM.04	The Associative Layer must support multitasking, in terms of maintaining multiple goals at the same time and switching between them, as well as interrupting a goal for reactive goals.
CM.05	The Associative Layer must support reactive behaviour, in terms of reacting on external events of visual or vocal type, limited to a set of events and rules, modelled in the procedure and/or a dedicated file.
CM.06	The Cognitive Layer must have a goal hierarchy, related to the goal hierarchy of the Associative Layer.
CM.07	The Cognitive Layer (or the Memory) must be able to translate signs into symbols
CM.08	The Cognitive Layer must be able to monitor the Associative Layer or the actions of the associative layer in the environment.
CM.09	The following Error Production Mechanisms should be implemented: <ul style="list-style-type: none"> • Learned Carelessness/Frequentia Simplification • Loss of information in Working Memory • Routine Capture • Attentional Capture/Selective Attention • Salience bias • Simple omission • Distraction due to external events • Action priming (Optional) • Recency bias (Optional)
CM.10	The VSP should support the creation and management of the crew

procedures via a dedicated Procedure Editor (PE). Requirements for this tool are:	
CM.10.1	PED should be executable in a standalone version
CM.10.2	PED shall permit to open and import procedures written in XML format according to a predefined DTD. The DTD shall incorporate the basic elements present in the procedure editor (GOMS elements)
CM.10.3	PED shall permit to export a procedure written in the procedure editor in an output xml file (according to DTD)
CM.10.4	PED shall permit to export a procedure written in the procedure editor in an output text file readable by the CM.
CM.10.5	The DTD file should be a separate file.
CM.10.6	PED should be able to import a list of objects (variables) from a .txt file to be used for filling the properties of "Percept", "Condition" and "Motor" actions (optional)
CM.10.7	PED should tag the relevant actor (e.g. PF, PNF).

In addition to these requirements, requirements and enhancements for the target system and the other modules (A/C module, environment model, etc) have been proposed. As these modules are equivalent to the modules in the PSP, these modules are developed and described in WP2 (see D2.4). The requirements and the implementation status for the scenario controller and the symbolic AHMI are described in section 6.1 and section 6.2 respectively.

4 Task Analysis and Task Modelling

As already described at the beginning of this document, the VSP in HUMAN will be used to predict crew behaviour with regard to errors. The predicted behaviour will be compared with real crew behaviour, which has been determined by flying the same scenarios with real pilots and cognitive pilots. To produce realistic scenarios the modellers of the cognitive architecture have to understand which tasks are executed by the crew in a particular situation and how the crew performs his tasks.

In order to acquire the normative behaviour for the cognitive crew, we use a hierarchical task analysis (HTA), which is integrated in our tool chain. For the simulation in the cognitive architecture, we need a procedure on a key-stroke level. The flight procedures that are used for training are often described in a textual form, and are thus very un-formal. To support the modelling of the input in a formal description, the tool chain in HUMAN includes two tools. The first one is AMBOSS. AMBOSS is used to model semi-formal task models for safety critical systems. This intermediate step eases the step from an un-formal description to a formal description. Output of AMBOSS is a semi-formal description of the tasks the crew has to perform. This output is input for the second tool, which is called PED. PED is

a procedure editor which supports the modeller in creating procedures as an enhanced GOMS model on the Keystroke Level.

The Keystroke Level Model was first time described by Card Moran and Newell in (Card et al. 1983). The Model based on a very detailed description of interaction and allows estimating time it takes to complete simple data input tasks using a computer and mouse. By using KLM-GOMS, analyst is able to find more efficient or better ways to complete a task simply by analyzing the steps required in the process and rearranging or eliminating unneeded steps (Kieras 1993). In HUMAN we are not going to use the Keystroke Level Model method, but we are going to investigate the procedure of a pilot on the same granular level.

After the procedures are created and validated by experts, we use these specifications of knowledge to run our cognitive architecture, which has a key position in the simulation platform.

As next we are going to present requirements given by our industrial partners regarding PED. Additionally we will describe the theoretical and practical background of our tool chain and give an overview of the state of the art.

4.1 Requirements from WP1 for PED

The industrial partners in HUMAN provided in the deliverable D1.3 requirements, which should be fulfilled by the Procedure editor implemented by OFFIS. The next table shows the list of requirements. This list is not closed and will be extended by all HUMAN partners if new requirements arise. Anyway most of the requirements are already fulfilled and communicated to the HUMAN consortium.

Req. No.	Description
R .01	PED should be executable in a standalone version
R .02	PED shall permit to open and import procedures written in XML format according to a predefined DTD. The DTD shall incorporate the basic elements present in the procedure editor (GOMS elements)
R .03	PED shall permit to export a procedure written in the procedure editor in an output xml file (according to DTD)
R .04	PED shall permit to export a procedure written in the procedure editor in an output text file readable by the CM.
R .05	The DTD file should be a separate file.
R .06	PED should be able to import a list of objects (variables) from a .txt file to be used for filling the properties of "Percept", "Condition" and "Motor" actions (optional)
R .07	PED should tag the relevant actor (e.g. PF, PNF).

4.2 Theoretical foundations for HUMAN Task Modelling

Hierarchical task models represent a well established approach for showing dependency between tasks and sub-tasks in a socio-technical system. The hierarchical structure of a task model is achieved by decomposition of the higher level tasks into lower level tasks (e.g. plans in hierarchical Task Analysis (HTA) (Shepherd, 1995)) in respect to the temporal relationship between the tasks (Bolognesi & Brinksma 1987). The result is a tree-like structure with temporal relations between the nodes. The primary objective is to model and analyze tasks, with visualization and a simulation part to understand what an actor (technical or human) does to reach a target. Because structures of task trees are easy to understand, it is a powerful method for communication between experts of different domains. This shows a potential benefit for this project, especially in the early phases of system or scenario development.

Task models document the order of and the rationale behind the planning and it structures the tasks which should be performed. This is useful for analyzing an existing socio-technical system or to start the design of a new one. A task model of a particular process can be helpful in detecting potential problems created by, for example, inadequate task order, disproportionate distribution of workloads between actors, or lack of time in critical phases of task execution (Giese et al 2008, Mistrzyk 2009, Basnyat 2006).

Task models have already been successfully used for supporting the constructive design of interactive systems. Their use as part of the product development process, however, is not yet well widespread between practitioners. There are various approaches that aim to specify tasks (Mori et al 2002, Baron et al 2006, Biere et al 1999, Barthet 1996). They differ in aspects such as the type of formalism they use, the type of knowledge they capture, and how they support the design and development of interactive systems.

Task modelling approaches such as (Mori et al 2002, Lu et al 2002, Stuart and Penn 2004, Baron et al 2006, Biere et al 1999) primarily concentrate on the hierarchical decomposition of tasks into subtasks and their relative ordering with temporal relations. Task models are typically used in the early phases of system design or as documentation method for the result of task analyses. The models are semi-formal in nature in the sense that they formally structure (hierarchy, temporal relations) informal elements (task descriptions). They have been used in system design (Mefisto 2000, Isolde 2005) or user interface design (GUITARE 1999, Szekely et al 1992), but are also used increasingly for the design and analysis of workplace situations (Mistrzyk and Redenius 2009) or professional project management (Stuart and Penn 2004).

GOMS models allow a more formal way of a task description. GOMS was developed by (Card et al. 1983) and is an engineering model for human performance to enable quantitative prediction. It is an engineering approach to task design. The original GOMS model, referred as CMN-GOMS, is the root of the family of GOMS models like GOMSL (GOMS Language) and CPM-GOMS (Critical Path Method GOMS).

The idea of GOMS is to reduce user's interaction with a computer to its elementary actions (these actions can be physical, cognitive or perceptual). Using these elementary actions as a framework, an expert is able to analyse an interface. As already mentioned there are several different GOMS variations which allow for different aspects of an interface to be accurately studied and predicted.

For the whole family of GOMS variants, the definitions of the major concepts are the same. Goals are what the user intends to accomplish. Operators are actions that are performed to get to the goal. Methods are sequences of operators that accomplish a goal. There can be more than one method available to accomplish a single goal. If this is the case then selection rules are used to describe when a user would select a certain method over the others. Typical GOMS analyses do not focus explicitly on the selection rules which are often ignored. For GOMS the essential concepts are methods which describe how goals actually are accomplished. Higher level methods describe task performance in terms of lower level methods. The lower level methods use task flow operators that act as constructors controlling the task execution. Although the granularity of GOMS models varies according to the purpose of the analysis, GOMS is mainly useful when decomposition is done at operational level. This circumstance will be used directly in our method.

4.3 HUMAN Task Modelling approach

The HUMAN approach starts with a textual description of flight scenarios and flight procedures. Parallel to the textual description the experts (e.g. pilots) develop a systematic description of the flight procedure in tabular form. The tabular form contains all actions needed to accomplish a certain flight task. The modeller can find in the table information about the duration and relation regarding to the tasks and actions, as well as information about devices which are used to perform a task, e.g. the mode of the AHMI. In addition it includes also information about possible deviations from the normative activities. Based on such information the modeller is able to develop a semi-formal hierarchical task model with AMBOSS which belong to the traditional task analysis and prepare adequate ground for the next step provide by PED which belongs more to the cognitive task analysis and describe task on the operational level. A directly step from the textual description to the formal task model on a operation level is too hard. The textual description doesn't contain enough structure and clearness needed to start with a cognitive task analysis. This is the main reason why we decided to use an intermediate step and integrated AMBOSS as a modelling environment. This step allows inspecting, verifying and validating the semiformal task models and prepares the already mentioned proper input for the further investigation.

For this step we are going to use the freely available software AMBOSS (AMBOSS 2009). With AMBOSS, the modeller is able to build a visual task model and to analyse it. The analysis can be distinguished in four ways: inspection, analysis,

verification and validation, as suggested by Martijn van Welie in (Martin van Welie et. al, 1998).

- **Inspection** means browsing through the data. A task model includes a lot of information which have to be understood. The graphical representation shows specific aspects of the data, such like who is performing which task, which message is send between which actors, which object is used where and by whom. AMBOSS provides several coherent and consistent representations of a model.
- Whereas inspection means merely “looking at” the **analysis** part is dealing with the question: “What is going on?” The goal is to understand the task world and to find the nature and causes of a problem. For example to mark all tasks which are linked to a particular rule or room can help to find dependencies between the tasks like temporal relations, interaction, and show how tight the cooperation in the socio-technical system is.
- **Verification** is defined on a more logical level. On this level, only a limited degree of a task model can be supported, due to the inherent lack of formal foundations for task models. The reason is the lack of existence of a task model to verify against. However it is possible and reasonable to have a look if the model satisfies certain domain independent constraints. It could be for example important to check if each task is related to a role, or if the sequence of performing a task is possible.
- **Validation** can be done only together with a domain expert. The expert has to check if the task model corresponds with the task world. AMBOSS provide a simulator for such checks. The domain actors can execute task by task and check the relations to rooms, object, rules, times, barriers and communication.

After the last step in the Analysis process, one can be quite sure that the structure of the semi formal model is valid, and can be used to develop a formal task model in a much more detailed form. The same method of analysing the task model can be used also in the next step of your approach, which describes tasks on an operational level. This step includes modelling of formal task models using our own modelling environment called PED. PED is developed at OFFIS. With this software the modeller is able to build GOMS models of the flight procedures on a key-stroke level. Figure 4-1 shows how the two modelling environments work together. The developed procedures can be exported into the cognitive architecture and there used for the simulations. In the next section we are going to describe modelling environments using in the HUMAN task modelling chain in detail.

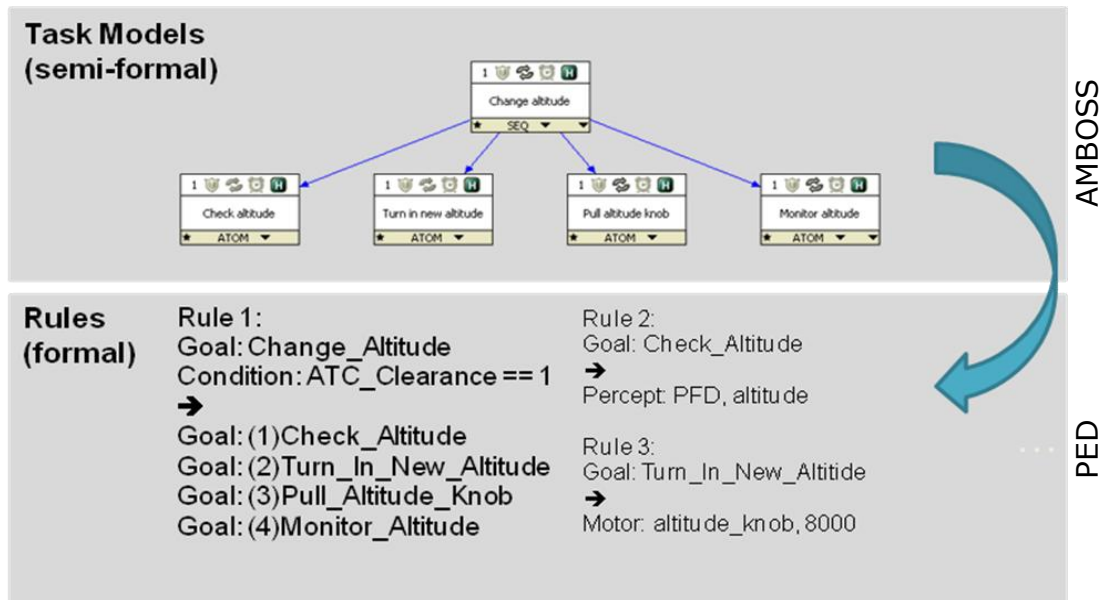


Figure 4-1: Transfer between the semi-formal and formal level of task modelling

4.4 Model of HUMAN Task Modelling tool chain

In this section the meta-models of the two tools used to model and analyse procedures and tasks are presented. In addition, a description of their functionalities is given.

4.4.1 Modelling environment for Semi-Formal Task Models

AMBOSS² is a free modelling environment, following the traditional approach of hierarchical task structures. AMBOSS supports hierarchical task modelling, especially for safety critical systems. The software provides the typical tree-based editing functions, such as editing a child node or a sibling node; apart from that, however, AMBOSS allows direct editing and manipulation of nodes and connections for simple structural manipulation tasks. AMBOSS allows describing tasks at different abstraction levels in a hierarchical manner, represented graphically in a tree-like format. Figure 4-2 shows the used notation. The modelling environment provides a set of temporal relations between the tasks such as:

- sequential: the subtasks are performed in a fixed sequence,
- serial: the subtasks are executed in an unsystematic sequence,
- parallel: in this relation the subtasks can start and end at random relation to each other,

² Homepage: <http://mci.cs.uni-paderborn.de/pg/amboss/>

- simultaneous: the subtasks start in an arbitrary sequence with the constraints that there must be a moment when all tasks are running simultaneously before any task can end,
- alternative: just one randomly selected subtask can be executed.

These are almost the same temporal relations that can be found in TOMBOLA (Uhr H. 1999) or in CTT (Mori et al 2002). A task node without any subtasks is automatically noted as an atomic task.

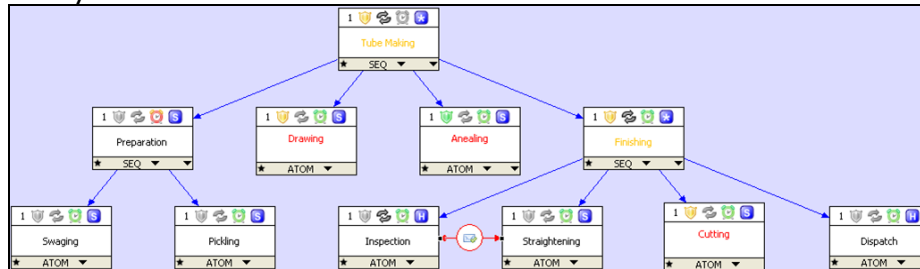


Figure 4-2: AMBOSS notation

The modelling environment has additional distinct views of a task model, which can be used for inspecting particular attributes of the tasks. For example, if an analyst likes to observe what kind of objects are manipulated in a system by a particular task, he can switch to the object view, take a look over the model, and analyze the dependencies between tasks and objects. It is also possible to review what kind of object is associated to a particular task, what kind of access rights (read or write) the task has and in which room the objects are located.

AMBOSS also enables the handling of topological aspects and relationships. To capture dependencies linked to the topology of a system, AMBOSS is able to build a relation between space and a task which is performed in a particular environment, for instance a cockpit of an airplane. Using this approach the modelling expert is able to model which part of a system a in a system a certain task is executed.

This concept helps to acquire a bigger picture of the tasks and their execution linking to a certain position of the work space. The user is able to specify, for example, that performing critical tasks is only allowed in a particular room. An expert can specify the relationship either from the point of view of a task or a room. Additionally, every object of the task model can be assigned to a room in a similar way as the task itself.

Analogous to other environments, like (Mori et al 2002), AMBOSS contains the concept of roles and actors. Basically there are three types of actors used: system, human and abstract. The user can specify additional actor subtypes, (such as engineer or manager), and instantiate each actors (such as Ms. Lee, the manager of marketing), to make it more concrete and fit the role to a particular socio-technical system. If an actor is linked to a task, his role can be also specified as being the person who is responsible for a particular task. This concept helps to organize areas of responsibility for each actor involved in a task model.

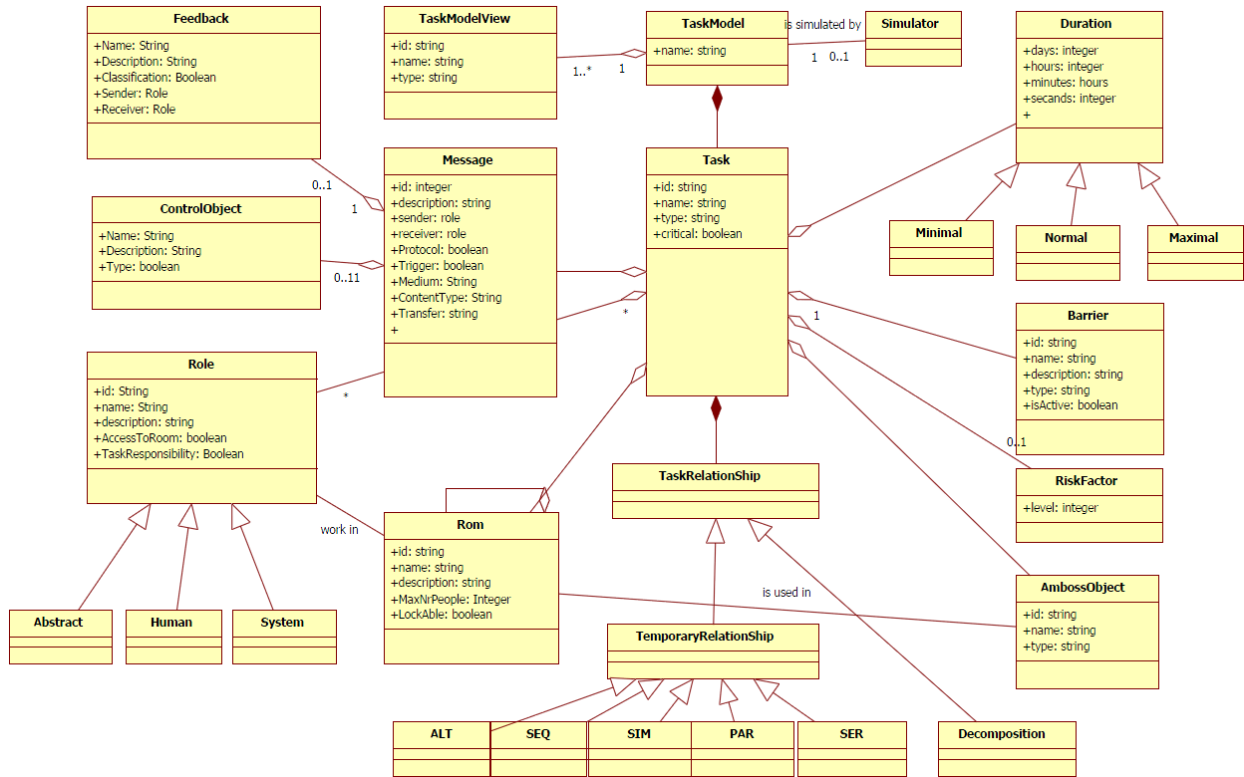


Figure 4-3: Meta model of the semi-formal modelling environment, AMBOSS

The main purpose of AMBOSS is to provide a hierarchical task modelling environment that supports developing and analyzing task models in safety critical domains. For safety purpose, AMBOSS contains the concept of barriers. This concept allows specifying parameters which help to protect human life and/or computer infrastructure. In a production process, barriers could be used to protect workers from hot temperatures. Barriers are a special kind of object which can be activated or deactivated by tasks. This can be directly “observed” during the simulation which is a part of the environment.

Similar to other modelling approaches, e.g. (Mori et al 2002, Uhr, H. 1999), AMBOSS is able to simulate a task model. The simulator presents the user exactly what happens in a task environment at a particular moment. A task model can be simulated by taking into account the task hierarchy, temporal relations providing the task execution order, and the communication flow showing the messages along with their parameters. Additionally, the user is able to observe the activation and deactivation of barriers, interdependency between tasks and their parameters. If discrepancies are detected, they can be immediately corrected. In that way, the user is able to validate his semi-formal task model.

4.4.2 Modelling environment for Formal Task Models

To develop an adequate low-level model for the cognitive architecture, the experts need to have a precise picture about the modelled cognitive system of the operator, as well as a highly precise description of the environment that the operator interacts with, and the task that has to be solved. The last section presented a solution for modelling semi-formal task models. In this section we describe our modelling environment for formal task models. To stay consistent to the hierarchical structure produced in the step before we use the XML output file of AMBOSS as framework for the procedure structure in the next step.

In order to support construction of the procedure models, we developed our own procedure editor, called PED. PED allows in an interactive and intuitive way to specify procedures using a procedure language. Our procedure language has been developed at OFFIS to allow description of normative activities in a general, context independent form. Based on the GOMS language (Goals, Operators, Methods and Selection Rules) the whole scenario can be organized in a tree-like structure of goals and sub-goals, methods and operators to achieve a certain goal and selection rules, if there is more than one method available to achieve a certain goal.

To get sufficient expressiveness of the procedure language we focus on three main topics specifying information which should be included:

1. Description of the situation in which this procedure is necessary, i.e. the possible conditions that trigger this procedure
2. Description, as detailed as possible, of the different steps of the procedure
3. Description of the different decisions that might be necessary in the procedure, and the different aspects on which the decision should be based.

As next we are going to present the semantic structure of the rule based language in details.

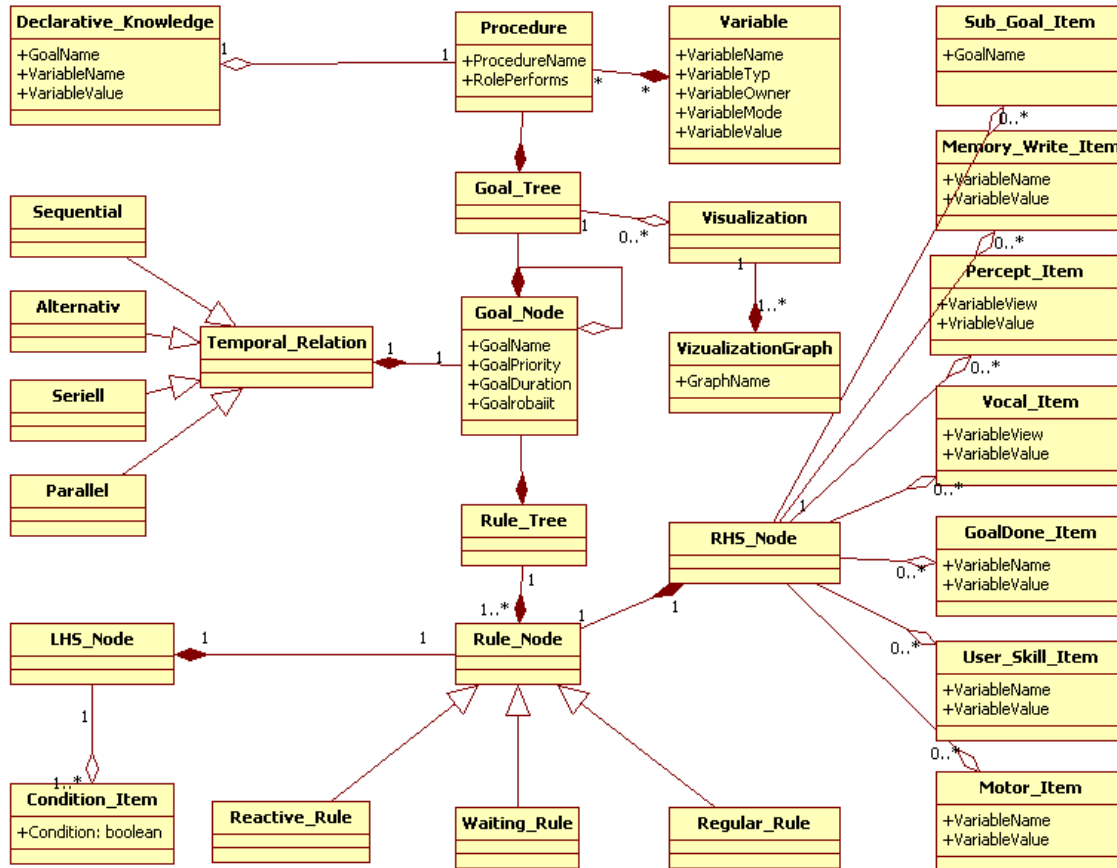


Figure 4-4: Meta-Model of PEDs Task Model

4.4.3 Rules based language

The syntactical structure of the rule-based language can be found in the appendix of this document in a Backus-Naur-Form. In this section the semantic part of the language is discussed. For better understanding how the procedure language is embedded into the modelling environment, Figure 4-4 shows the meta-model of PED. An example of a Rule and its parts is shown in Figure 4-5.

To build a procedure, a modeller has to describe a set of rules which define in a formal way the task to be performed. Each rule contains a left and right side. The **left-hand** side defines the conditional element which contains the Goal- and the State-Part. Each rule is responsible for exactly one goal, and can be chosen if the Boolean expression of a condition, defined over system variables (e.g. current mode) or environmental variables (e.g. current altitude), is true.

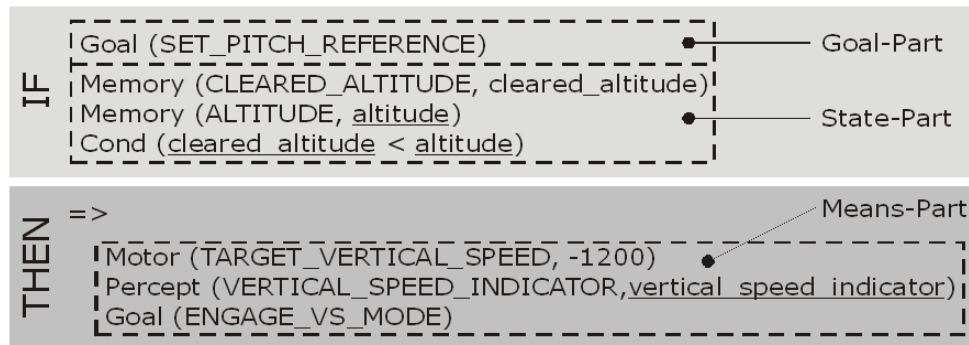


Figure 4-5: Rule with the Goal-,State- und Means-Part

The **right-hand** side specifies a method and contains the Means-Part of a rule. A method is a sequence of different types of operators. Each operator is described in detail in the next section. Additionally, the language is able to specify different temporal relations giving the order of the sub-goals. The temporal operator is an attribute of a goal and gives the order of all sub goals which are directly linked to the goal. (In a tree view this is a father-son relation.) At the moment there are three temporal relations included in the language.

1. A **Sequential temporal relation** gives a strict sequential order to all sub goals and actions which have to be executed after each other.
2. An **Alternative temporal relation** means that only one of a set of alternatives may actually be chosen and executed. The actions in the set of alternatives exclude each other.
3. A **Serial temporal relation** means that the sub-tasks can be done fully independent of each other. If this relation is used, the modeller should also give information about the priority of each action or sub goal.

Above all, the language is able to specify goals in more detailed way and define their priority and probability. The **probability** parameter should be used together with the alternative temporary relation. With this concept the user is able to specify how high or low is the reachability (likelihood) of a sub goal. The **priority** parameter should only be used with the serial temporal relation. Using this concept, the modeller is able to specify how high or low is the priority of a particular sub goal.

Operators are the functional entities of the procedure model. They are distinguished in:

- **Percept** operators serve to acquire information about the environment via the displays.
- **Motor** operator represents physical pilot actions, such as pressing a button or entering data into a device (e.g. alerter).
- **Memory write** operator stores items in short-term memory
- **Memory read** operator retrieves items in short-term memory

- **Vocal** operator symbolises communication (in HUMAN there is communication between PF,PM,ATC)
- A **Skill** operator interacts with the autonomous layer, and starts and stops the skills of the user.
- The **Goal done** operator “states” that a goal is completed and removes it from the goal agenda.

4.4.4 Implementation Status of the HUMAN Formal Task Modelling

The modelling environment PED is already running as a stand-alone software. The user is able to build a model in an intuitively way using the user interface. Figure 4-6 shows a screenshot of PED with different views. We use the concept of different views to provide a systematic view of different parameters of a procedure. At the moment PED provides a Goal-, Rule- and Property-View. Furthermore PED offers a dual procedure editing mode, allowing to model different actors at the same time, e.g. pilot flying and pilot monitoring. In this case the goal and rule view of pilot flying are positioned on the left hand side, goal and rule view of pilot monitoring are positioned laterally reversed on the right hand side.

In addition, the user can use a separate view integrated in PED to define all needed variables of a procedure. In this view the user creates, changes and deletes variables for a model. Each variable has got a name, type and mode. There are four possibilities to choose from for the type of a variable: String, Integer, Float and Double. Each variable has to be related to a mode. At the moment there are three modes:

- **Out** mode, is a mode for variables which an actor can change with his actions. It can be used for motor or vocal actions.
- **In** mode, is a mode for variables which an actor can percept. These variables can be used in percept actions.
- **Internal** mode is a mode for variables which an actor has only in his working memory, and cannot perceived in the environment. It could be used for memory write or memory read actions and in conditions.

The variables can be assigned to the pilot flying, pilot monitoring or both actors at the same time. Beyond that the user is able to specify also the declarative knowledge for an actor (in HUMAN pilot flying and pilot monitoring). Declarative knowledge is mainly meant to be long-term knowledge that is learned, e.g. the knowledge that the autopilot has to be in a certain mode when the pilot intends to perform a certain task. The declarative knowledge is visualized in a tabular form in that the modeller is able to specify the value of variables for each goal.

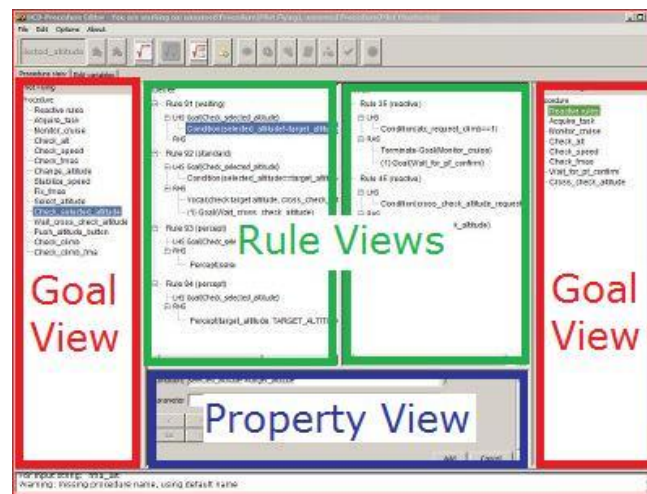


Figure 4-6: Views of Procedure Editor with dual procedure view mode

In addition to the different views, PED offers also visualization of a procedure as a graph. To get an optimal graphical presentation of important information, the user can customize the visualization and choose which parameter of a procedure should be shown. Figure 4-7 shows a graphical representation of a procedure in a horizontal layout. Using this graphical visualisation, the user can better understand the structure of a procedure with a focus on particular issues. Such information supports the user but also an analyst during the analysis of a model, as suggested already in section 4.3.

All requirements for PED, as defined in D1.3 and D3.1 have been implemented and accepted by the industrial partners.

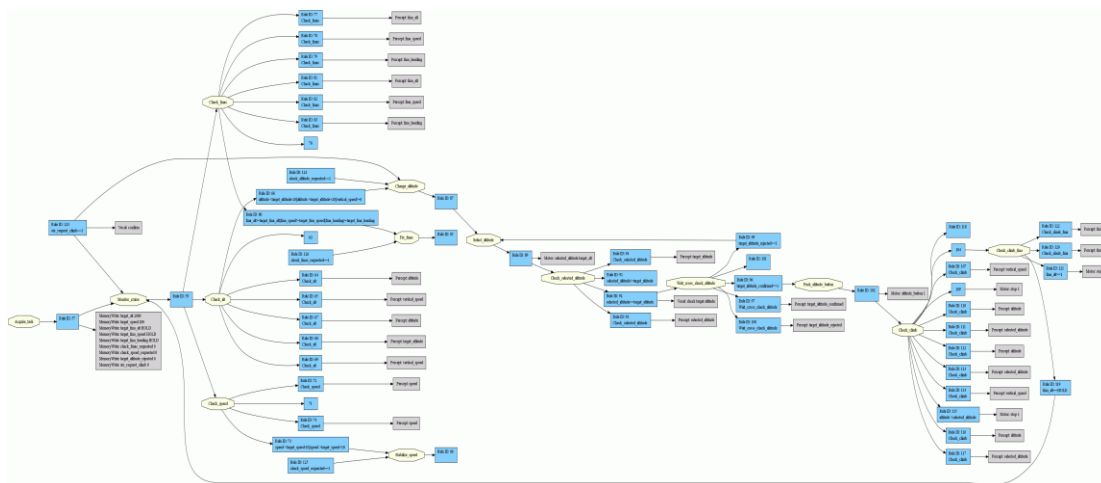


Figure 4-7: Graphical representation of a procedure

5 Cognitive Architecture

As written in the introduction, the cognitive architecture interprets the normative crew behaviour that has been described in the previous section. In HUMAN we use CASCaS (**C**ognitive **A**rchitecture for **S**afety **C**ritical **T**ask **S**imulation), a cognitive architecture whose development has been started at OFFIS, in the 6th EU framework project "ISAAC". In the Human Error Analysis (HEA) theme of ISAAC, a methodology for the safety analysis of aircrafts with regard to human factors has been developed. CASCaS has been extended in HUMAN, so that more errors can be predicted. According to the results of ISAAC, and the Error Types and Error Production Mechanisms that are under investigation in HUMAN, several requirements for the cognitive architecture have been proposed in WP1, see the following table:

Req. No.	Description
CM.01	The VSP must be able to simulate a crew, consisting of PF and PM. Therefore the platform must support communication between PF and PM, restricted to simulated verbal communication.
CM.02	The VSP must be able to interpret crew (PF, PNF) normative procedures.
CM.03	The perception of the CM must be sophisticated, in terms of a restricted visual field with focus, as well as eye-movements. Perception must support the detection of certain modelled events, e.g. warning lights or sounds, which trigger reactive rules.
CM.04	The Associative Layer must support multitasking, in terms of maintaining multiple goals at the same time and switching between them, as well as interrupting a goal for reactive goals.
CM.05	The Associative Layer must support reactive behaviour, in terms of reacting on external events of visual or vocal type, limited to a set of events and rules, modelled in the procedure and/or a dedicated file.
CM.06	The Cognitive Layer must have a goal hierarchy, related to the goal hierarchy of the Associative Layer.
CM.07	The Cognitive Layer (or the Memory) must be able to translate signs into symbols
CM.08	The Cognitive Layer must be able to monitor the Associative Layer or the actions of the associative layer in the environment.
CM.09	The following Error Production Mechanisms should be implemented: <ul style="list-style-type: none"> • Learned Carelessness/Frequent Simplification • Loss of information in Working Memory • Routine Capture • Attentional Capture/Selective Attention • Salience bias (• Simple omission • Distraction due to external events • Action priming (Optional) • Recency bias (Optional)

In order to fulfil these requirements, the first step is to extend the theoretical foundation on which the architecture is based (see section 5.1). As a second step, several basic capabilities have been proposed, that should fulfil the requirements, (see section 5.2)

5.1 Theoretical Foundation for the Cognitive Architecture

Rasmussen developed a theory of cognition in his books and papers on human error analysis (see Rasmussen, J.; 1986, Rasmussen, J.; 1987). According to his theory, human cognition consists of three levels, the skill-based level as the lowest level, the rule-based level and the knowledge-based level as the highest levels, see Figure 5-1.

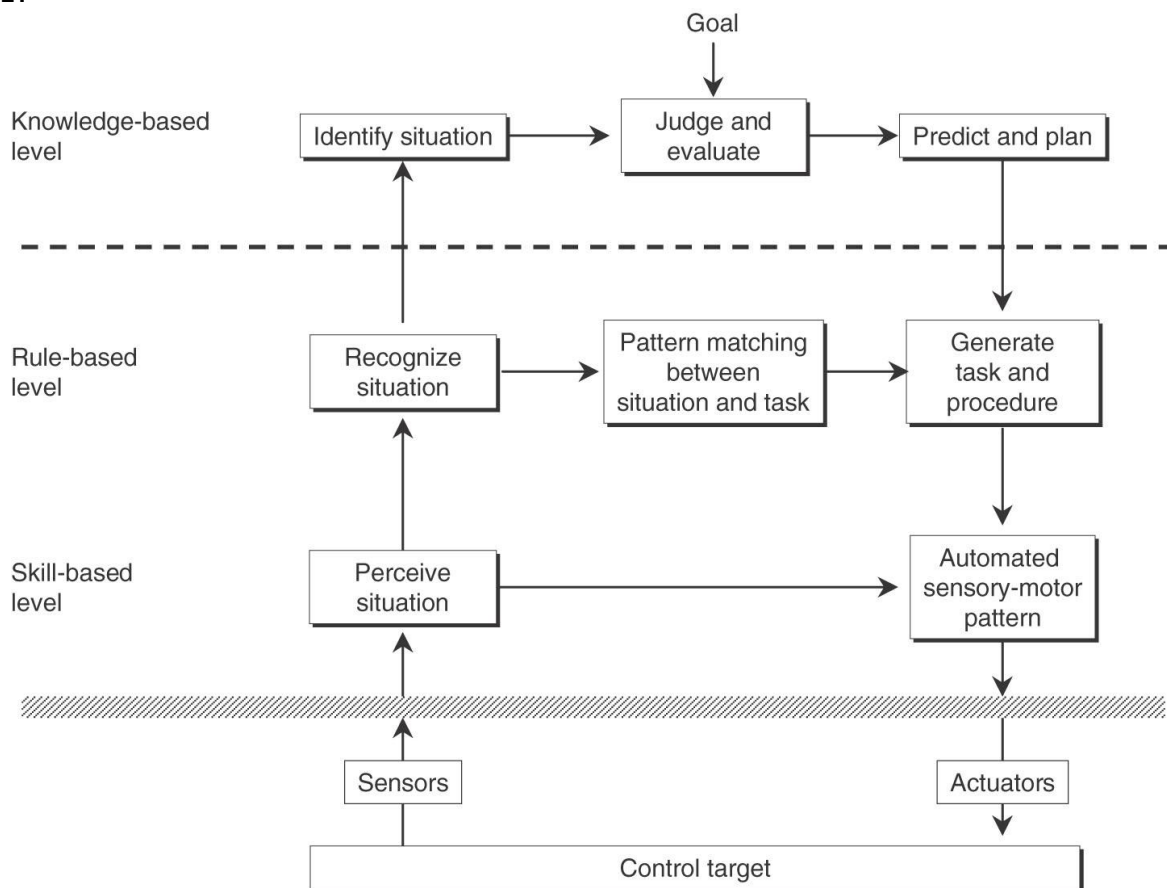


Figure 5-1: Cognitive Process from Rasmussen

On the skill-based level, human act highly automatically, without “thinking” about it, and people cannot describe how they act. This is for example steering a car or a plane on the ground (in terms of holding the car/plane on the lane). On the rule-based level, humans use procedures for familiar procedures to act. This is used in situations that are often performed, e.g. using a system according to a well learned

procedure, like controlling the autopilot. When it comes to unfamiliar situations or situations where non-procedural cognition is needed, humans use the knowledge-based level, e.g. for decision making. In general, people tend to stay on the lower level as long as possible, as this is normally faster and easier (i.e., it uses less cognitive resources). This becomes clearer, when one considers the three levels of control loops on the control target and counts the number of steps that have to be performed at each level, e.g. at the skill-based level, only two steps are needed to close such a loop: "Perceive situation" and then "Automated sensory-motor pattern". If interaction with the control target fails at this level, and the human switches to the rule-based level, 5 steps are now needed: "Perceive situation" which then leads to "Recognize situation", followed by "Pattern matching between situation and task", "Generate task and procedure" and finally "Automated sensory-motor pattern". Correspondingly, at the knowledge-based level, 7 steps are necessary. These steps are thus performed in loops, i.e. the steps are performed again and again. Parallelism between the levels is possible (and usually the rule), e.g. while acting on the skill-based level, it is also possible to execute a procedure on the rule-based level, with the constraint of limited resources (e.g. hands, eyes) when they are shared or requested by concurrent loops. In addition to this, multi-tasking is also possible at the rule-based level, e.g. humans can work on two procedures in parallel (i.e. switch between procedures when necessary, or even perform the two of them in full parallelism, provided they do not require the same resources).

In addition to the three levels, Rasmussen also assigns a type of information to each level. Information is categorised into signals, signs and symbols. At the lowest, skilled-based level, signals represent the information as it has been perceived, e.g. altitude is 200 feet. Signals can then be enriched with further contextual information, e.g. altitude < 1000 feet, and transformed into signs, to be used at the rule-based level. These signs can then be associated to semantic information and general knowledge and transformed into symbols, to be used at the knowledge-based level. The low altitude situation detected at the rule-based level could for example be categorised as "dangerous". Thus, each level is associated with its own type of information (skill-based → signals, rule-based → signs, knowledge-based → symbols). The idea is, that each transformation from a lower level to a higher level is accompanied by a translation of the information to a higher level by adding more contextual information, e.g. when the knowledge-based level is activated, the signs are translated into symbols. All this has the price of using some cognitive resources. Hence, the lower the level a loop is closed, the less cognitive resources it uses.

5.2 Model of the Cognitive Architecture

Based on the ideas of Rasmussen, we refined the cognitive architecture from ISAAC. The old architecture was able to interpret procedures, which is comparable to the

rule-based layer from Rasmussen, but non-procedural cognition was not possible, e.g. decision-making, as well as skill-based behaviour. These levels are now added to the architecture, as well as the concept of signals, signs and symbols. This results in the architecture depicted in Figure 5-2. In the following, we name the skill-based level of Rasmussen “Autonomous Layer”, the rule-based level “Associative Layer” and the knowledge-based layer will be the “Cognitive Layer”. These terms are from ACT-R, see Anderson 2000.

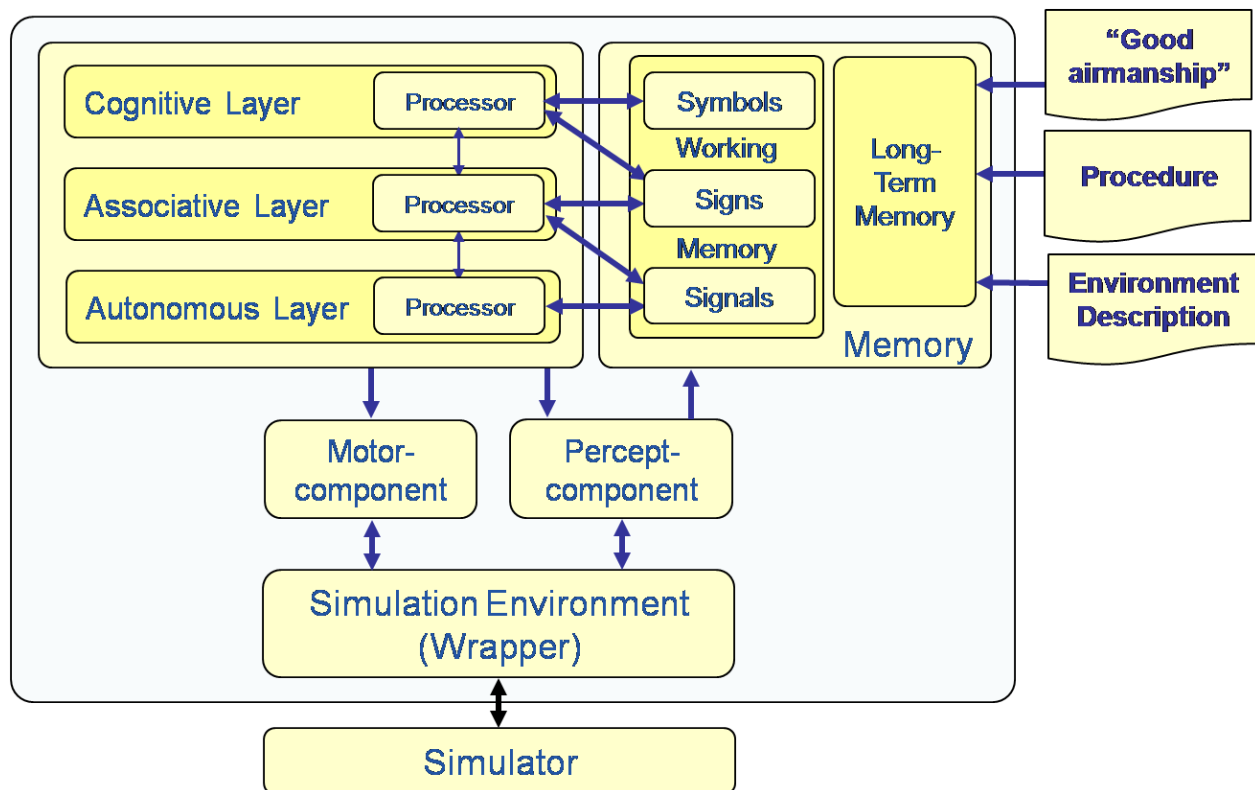


Figure 5-2: Cognitive Architecture in HUMAN

5.2.1 Cognitive Architecture

5.2.1.1 Crew Coordination

In aviation aircraft are flown in crews because of security reasons, and to share effort in the cockpit. The second crew member (pilot monitoring, PM, also known as pilot non flying, PNF) is used as a security net for the pilot who is actually flying the aircraft (pilot flying, PF). The PM cross checks the actions of the PF, confirms some of the PFs actions, reads check-lists, and has also his own tasks, e.g. communication with ATC. In order to be realistic, the VSP therefore has to consist of a crew, in terms of two separate cognitive architectures, one with the model of the

PF, and one with the model of the PM. In a real cockpit, there is communication between the PF and the PM, e.g. when a cross check is performed, or a checklist executed. This communication is either verbal, or visual, e.g. the PM could raise his thumb to give his OK. In the following, we assume that we have only verbal communication, in order to simplify the communication model, which is described in the next section. In section 5.2.1.1.2, we discuss the current implementation of the crew coordination. We do not consider visual communication, because this complex topic cannot be modelled in the framework of the project. In addition, data acquisition for visual communication is hard to achieve, because it has to be done via video analysis, which needs a lot of effort, and cannot be automated.

5.2.1.1.1 Communication Model for Crew Interaction

To investigate the crew behaviour in an adequate way, there is a need to integrate communication into the cognitive model. This will be done to understand what kind of information the crew deal with and how the actors communicate. In addition, communication failures in the aeronautic area contribute to a significant amount of accidents (Johnson 2004, Orasanu 1993).

There has been much research on communication and communication errors especially regarding cockpit communication (Sexton & Helmreich 2000). In order to explain errors during communication, suitable communication models have to be developed or assumed as fundament for the research. The model taken should be strong enough to allow explaining the sources and mechanisms of communication errors. As a first draft for a communication model in HUMAN we use one of the most well known communication models is the communication model of Shannon-Weaver; see (Shannon & Weaver 1949). Figure 5-3 shows the symbolical representation of this model. In the last decades, this model has been basis for many other communication models and has widely been extended. For the first cycle we use this simple model as a starting point. For the second cycle, we will re-investigate the communication model, and based to the experimental data we may choose another communication model, e.g. from cognitive science, that fits better to our needs. As we are not going to model communication errors in the first cycle, this decision will not have an influence on the first cycle of HUMAN. If the data in the experiments show a significant number of communication errors, the communication model can be easily exchanged.

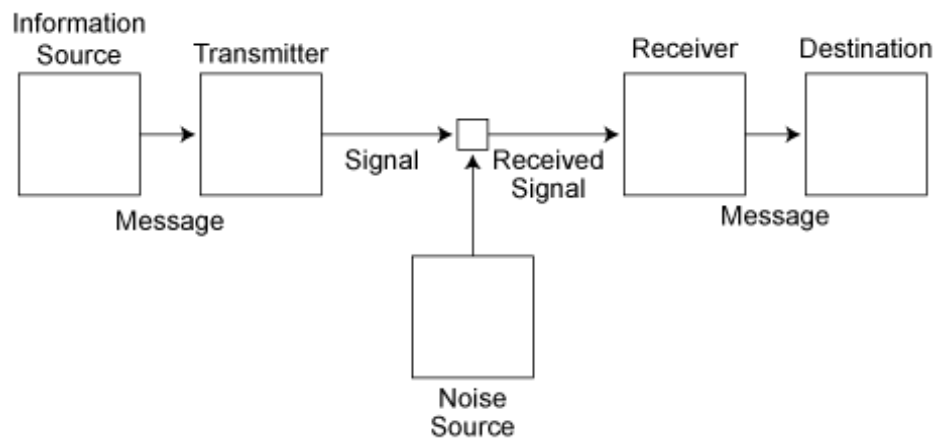


Figure 5-3: Schematic diagram of a general communication system (Shannon & Weaver, 1949)

That model comes from the information theory and proposes that all communication has to include six elements:

- source
- encoder
- message
- channel
- decoder
- receiver

A selected **message** includes **information** and may according to this model consist of written or spoken words, or of pictures, music, etc. In the case of pilots there are spoken words and written information on the e.g. primary display.

The **transmitter** changes this message into the signal which is actually sent over the communication **channel** from the transmitter to the **receiver**. In other words the transmitter (encoder) is responsible for taking the ideas of the source and putting them into code. It expresses the purpose of the **source** in form of a message. In a cockpit the channel is for example the air or the headphones. In the case of telephony, the channel is a wire, the signal a varying electrical current on this wire; the transmitter is the set of devices (telephone transmitter, etc.) which change the sound pressure of the voice into the varying electrical current. In oral speech, the information source is the brain of an actor, the transmitter is the voice mechanism producing the varying sound pressure which is transmitted through the air, the receiver is a sort of inverse transmitter, changing the transmitted signal back into a message, and handing this message on to the destination. In a later versions of communication model of Shannon and Weaver the receiver has got also the role of **transmitter**. Such models can also describe **feedback**. During a

communication process certain things can be added or excluded from a signal. If these things were not intended by the information source, it means that such phenomenon can destroy partly or the whole communication process. It could be for example errors in transmission, dissections of sound (in telephony, for example) or distortions in shape or shading of picture (television).

Weaver added three levels of communication errors to the mathematical description of the technical communication process from Shannon. He described the following levels of errors (cited from Weaver, 1949):

- A: How accurately can the symbols of communication be transmitted? (The technical problem.)
- B: How precisely do the transmitted symbols convey the desired meaning? (The semantic problem.)
- C: How effectively does the received meaning affect conduct in the desired way? (The effectiveness problem.)

In HUMAN, in the first version of the VSP, we will not model errors based on communication phenomena (cf. Deliverable D1.2). In order to be prepared to add this for the second cycle, we already define the communication model. This will allow us to add new error types or production mechanisms later, e.g. noise in the transmission, etc.

5.2.1.1.2 Symbolic Crew Communication

As written in the introduction for the crew coordination, the VSP will consist of two instances of the cognitive architecture. The first instance will run the model of the PF, and the other will perform the duties of the PM. As the crew has to work cooperatively, communication is needed. In the following we will describe the symbolic communication between the cognitive models.

Symbolic communication means, that the different cognitive architectures exchange messages between each other via the datapool. In terms of Shannon-Weaver, the datapool and the Simulation Environment (Wrapper, see Figure 5-2) are the channel for transmission. The information source is either the Associative or the Cognitive Layer, which can fire verbal actions. These actions are received by the speech module in the motor component, which then calculates the speech time and sends datapool messages accordingly. Thus the speech component is the transmitter of the message. After the datapool message has been received from the other cognitive architecture, the auditive component receives the message and creates memory entries in the working memory accordingly. See Figure 5-4 for a representation of the involved components.

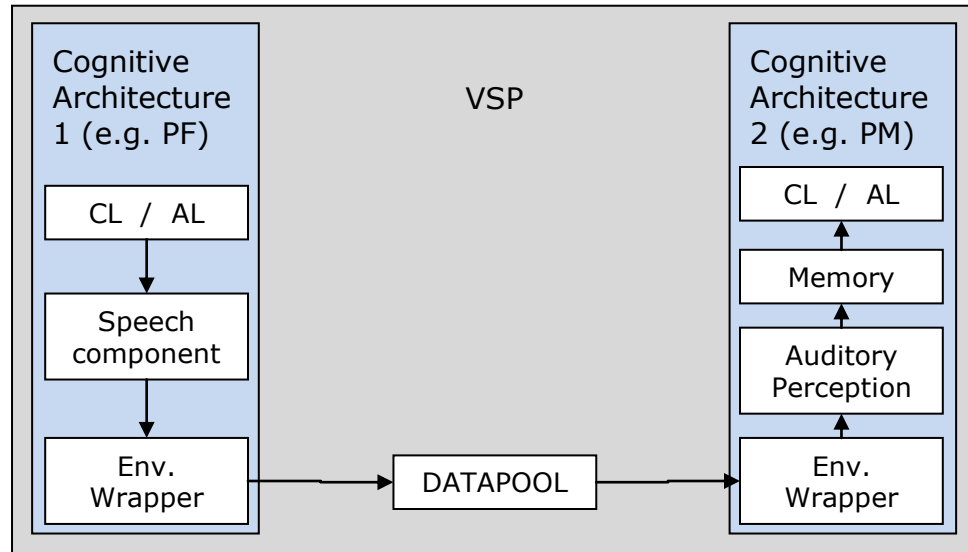


Figure 5-4: Schematic representation of Symbolic Communication

In order to use the verbal communication in the normative behaviour, a new action has been added to the procedure language, the “vocal” action:

Vocal (<spoken text>, <message>, <receiver>)

- <spoken text> is the text that is actually spoken by the architecture. This text is mainly used to determine how long the architecture has to speak.
- <message> is a list of memory writes, in the form <variable>=<value>(&& <variable>=<value>)*, i.e. “CLB_CLEARANCE=1 && TARGET_ALT=30000”. These entries are written to the memory as a signal, and can be used as a sign to trigger reactive rules or in normal rules (see also section 5.2.2.1 for description on reactive behaviour), or can be transferred into a new symbol on the cognitive layer.
- <receiver> determines the target of the vocal message. In HUMAN this will always be the other pilot, as there will be no vocal communication with ATC.

The main objective of the speech component is to determine how long the vocal action is. For this, we use an estimated speech rate of 190 words per minute for calculating the duration of the spoken text. This information is used to send two messages to the datapool. First a starting message, that signals to the receiver that the architecture starts to speak, and after the calculated time elapsed a stop message that signals the receiver the end of the speech. The stop message also includes the <message> part of the vocal action that is written into the working memory of the receiver. The separation in two different messages is necessary, because the receiver is normally not able to determine how long another person is speaking. So it is not realistic to send one message with the duration of the communication to the receiver. In the auditory component, these messages are processed accordingly.

5.2.1.2 Visual Perception

The visual component of the cognitive architecture after ISAAC was rather primitive. Mainly no distinction between focus and visual field existed, thus the percept component was able to perceive everything in the environment in a fixed time. The new visual component, distinguishes between focus and visual field, and thus allows to model eye-movements, and a more sophisticated visual behaviour. In addition to this, we model also visual bottom-up attention, or selective attention. Visual bottom-up attention describes the effect that certain salient cues in the environment, as for example a flashing or moving item, automatically changes the attention to this item (in terms of an eye-movement to the item), see (Folk & Remington, 1994). Recent studies have shown however that those automatic shifts are not always initiated. Two main causes have been identified for this: top-down attention prevents the shift of attention (Connor, Egeth & Yantis, 2004; Yantis & Jonides, 1990), or the display context undermines the effect (Nicolic, Orr & Sarter, 2004).

In the following, the concepts for the new visual perception are described. Figure 5-5 shows the architecture of CASCaS (cf. also Figure 5-2), with more details on the percept and motor components. In this section, the percept component is described in more detail. Section 5.2.1.3 will describe the motor components.

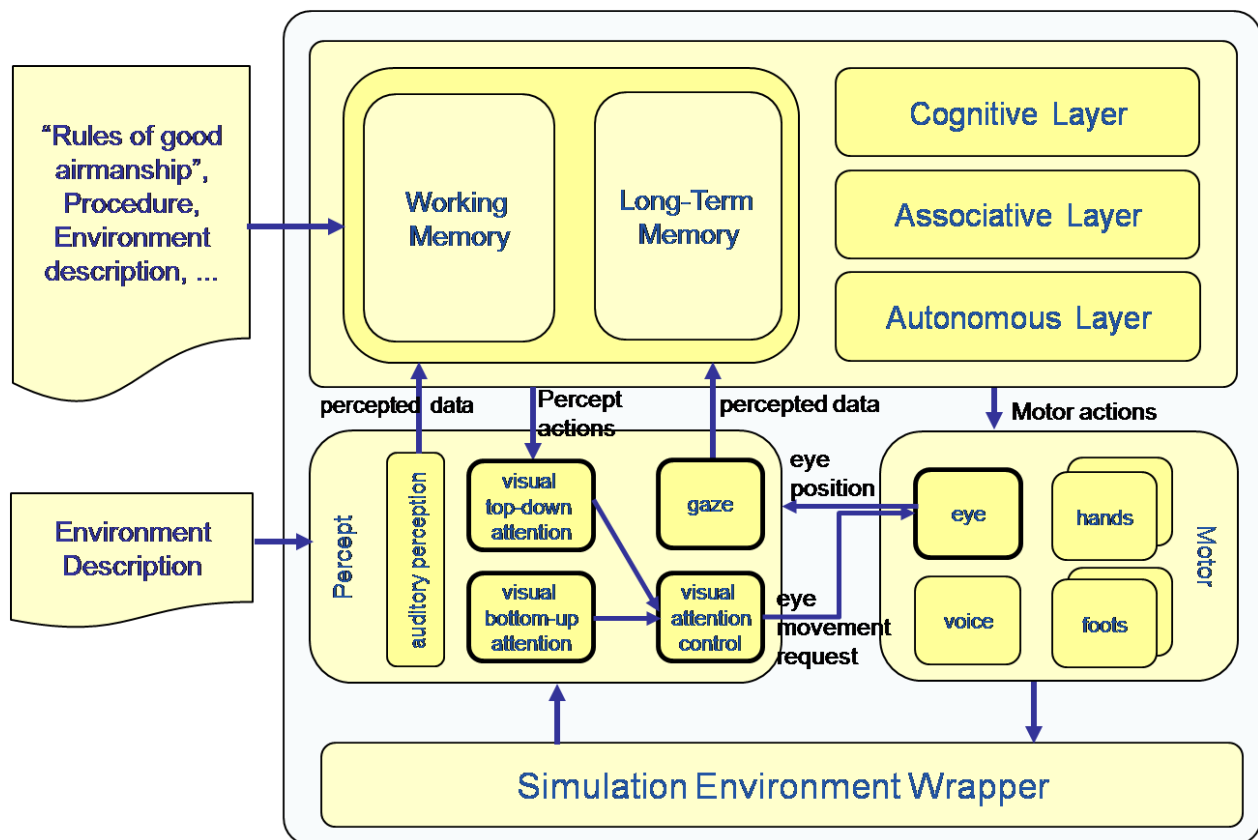


Figure 5-5: Architecture with extended Percept and Motor components

The percept component consists of five sub-components, one component for auditory perception, and four components for visual perception:

- The auditory perception component is needed for the crew coordination and interaction with cockpit systems. This component hears sounds and voice messages, as described in section 5.2.1.1.2.
- The visual top-down attention (cf. (Connor, Egeth, Yantis 2004)) manages all requests of shifts of visual attention that comes from one of the three layers (Cognitive Layer, Associative Layer or Autonomous Layer), as described in section 5.2.1.2.1.
- The visual bottom-up attention (cf. (Connor, Egeth, Yantis 2004)) manages vision in the visual field, as described in section 5.2.1.2.2.
- The visual attention control component (cf. (Connor, Egeth, Yantis 2004)) solves conflicting requests for shift of attention, e.g. from the top-down and the bottom-up component. Currently it gives top-down requests priority before bottom-up attention. This component controls the eye movements in the motor component.
- The gaze component symbolically retrieves data from the environment, according to the current eye position, and stores it in the working memory.

An important input for the percept component is the “Environment Description”, in the following called topology. The topology describes in which environment the virtual human currently is, in terms of area of interests (AOIs). Each AOI depicts a shape (incl. colour, size, and position), as well as information on how to manipulate the instrument (see description in motor component in section 5.2.1.3).

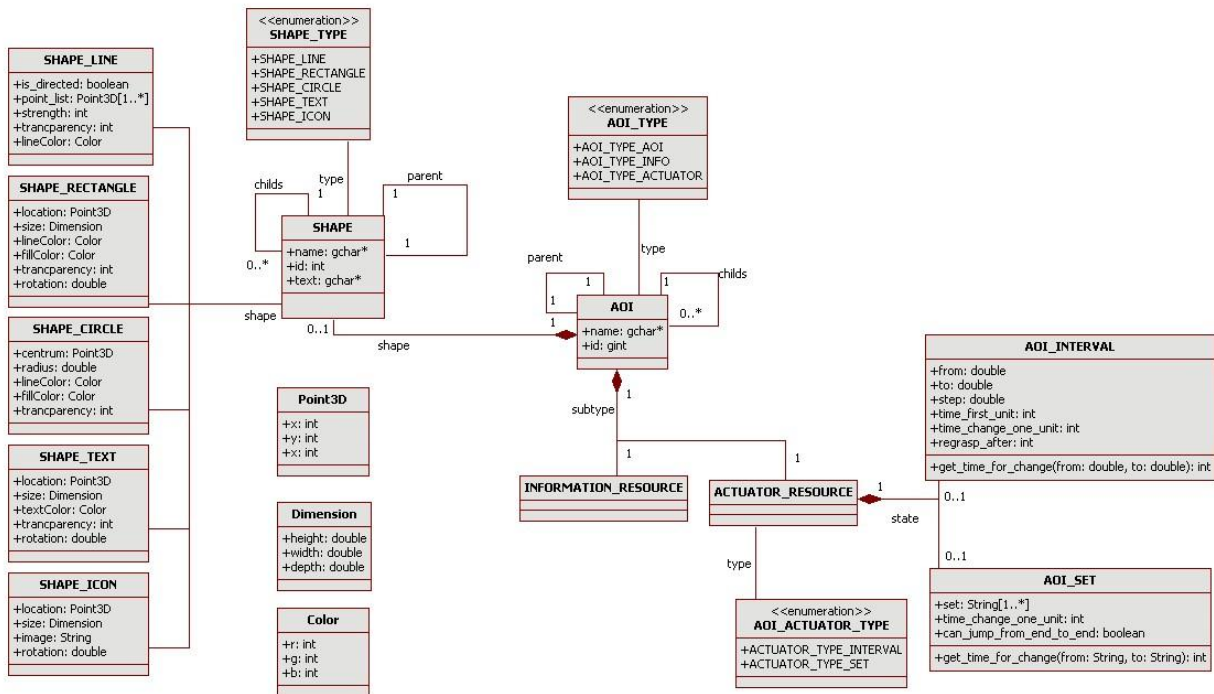


Figure 5-6: Topology of the Environment

The contained position information is mainly used to determine the angle between the instruments, for the movements of eyes and hands in the motor component, but also to determine which instrument is currently in focus, or in the visual field. The next section describes the visual top-down attention and the gaze component. The visual attention controller simply decides which shift of attention is performed in the next step (currently there is a priority of top-down on bottom-up), thus bottom-up is maybe delayed. In future version of CASCAs this may be extended, e.g. we can think about letting the associative layer or the cognitive layer decide what to do. This could be for example also interesting for the Saliency Bias. Last but not least, section 5.2.1.2.2 describes the bottom-up visual attention component, and its related EPM selective attention.

5.2.1.2.1 Visual Top-Down Attention and Gaze Component

The visual top-down attention maintains the percept actions, as requested by the AL or CL, and if multiple percept's have been fired in one step, it executes them sequentially.

Figure 5-7 shows the size of the focus and the visual field that has been implemented:

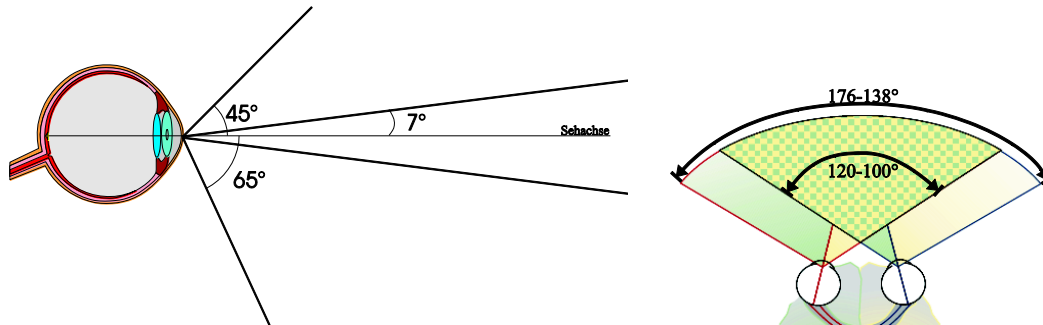


Figure 5-7: Expansion of horizontal and vertical visual field

It is planned to have a focus of 2 degree, and a visual field of 170 degree horizontal and 110 degree vertical.

The gaze component retrieves the values that are in focus (currently one single instrument). As long as the gaze is not moved to another instrument, the signal of the instrument is constantly updated in working memory. The minimum time of a fixation is 200ms (cf. (Fischer, 1999)).

For future implementations, it is planned that the gaze component successively retrieve and stores additional information, like position, size, shape and colour in the memory.

5.2.1.2.2 Selective Attention

In HUMAN, we define bottom-up visual attention as automatic shifts of attention due to static or dynamic discontinuities, and Selective Attention as an EPM, which causes a failure of bottom-up visual attention under some circumstances. Static discontinuities are differences in colour, shape or brightness, e.g. a red circle in a set of green circles, or a circle in a set of rectangles. In contrast to this, dynamic discontinuities are changes over time, e.g. a flashing or moving object. In the following, we will refer to the occurrence of a dynamic discontinuity as *event*. If such an event occurs in the visual field, this can result in a shift of visual attention (Yantis and Jonides 1990). This effect is often used by display designers, to capture the attention of users to a certain important change, e.g. the flight mode annunciations of the Primary Flight Display in a modern glass cockpit is flashing for about 10 seconds, if a mode changes. A mode change is an important event that a pilot has to be aware of. Unfortunately, recent studies, e.g. by Mumaw, et al. 2004, showed that bottom-up visual attention is not always working. One reason is that

top-down visual attention can prevent a shift of attention, e.g. when a pilot is very focused on acquiring certain information on another display. A second reason could be that the event is outside the visual field at all, e.g. because the pilot is looking down for a longer time. The third reason is that the event is missed because of interference with other discontinuities. Nikolic, Orr and Sarter called this "display context" (Nikolic, Orr, and Sarter 2004): If the event occurs on a display where the neighbourhood is very colourful or dynamic, the probability of recognising the event is reduced. We modelled this as a two step process. In a first step, for all events it is determined if the event occurs in the visual field. If it is in the visual field, a probabilistic choice is done, in order to determine if the event is recognised or not. For the probabilistic choice, we use the data given by (Nikolic, Orr and Sarter 2004): If there are dynamic instruments in a neighbourhood of 15 degree around the event, then the probability is 0.65, else it is 0.95.

For HUMAN, we will simplify this, in order to reduce the implementation effort. The only visual events that are of interest for HUMAN, are the mode changes of the FMA. As we know, that this events occur in a dynamic neighbourhood, the probability for the probabilistic choice can be always 0.65. If the concept seems to be valid after the first cycle, the implementation can be extended for a more general set of events.

The visual field of a human being is oval and has a horizontal extend of 170 degrees, and 110 degrees vertically. In order to make the visual field computational by our model, we abstract the oval visual field, by partitioning the visual field in four sectors, with the focus in the middle. Figure 5-8 shows the segmentation of the visual field. "I" depicts an example instrument, and " I_M " the angle between the positive x-axis and the instrument. In the example in Figure 5-8, the instrument I is in the left sector.

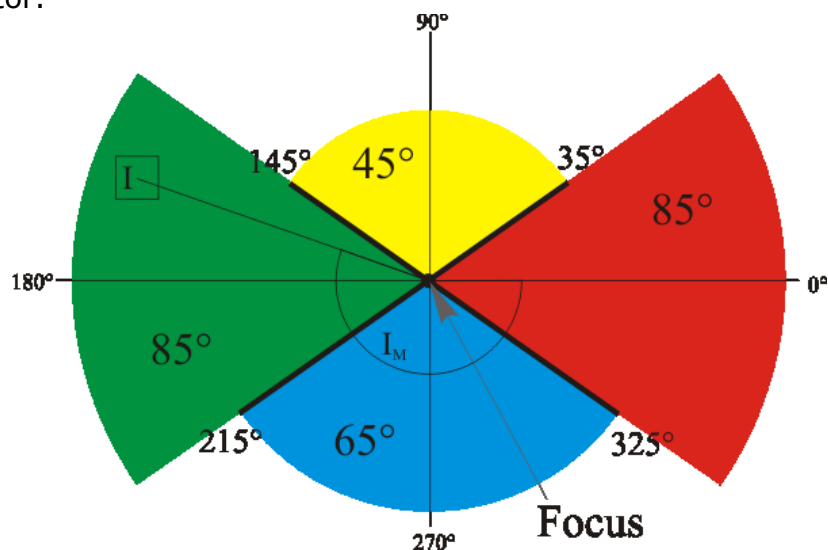


Figure 5-8: Segmentation of Visual Field

In order to be in the visual field, “ I_M ” has to be within certain limits:

- Top segment (yellow): $I_M \in]35,145]$, max distance to focus: 45 degree
- Left segment (green): $I_M \in]145,215]$, max dist. to focus: 85 degrees
- Bottom segment (cyan): $I_M \in]215,325]$, max dist. to focus: 65 degrees
- Right segment (red): $I_M \in]325,0] \vee [0,35]$, max dist. to focus: 85 degrees

An event has to be within these limits, in order to be in the visual field.

5.2.1.3 Motor

In the old version of CASCaS, after ISAAC, the motor component was very simple. It consisted only of one hand, which needed always a fixed time to adjust a value. We extended the motor component with a richer set of actions, as well as more resources. As depicted in Figure 5-5, the motor component is decomposed into four sub-components or resources:

- The *eye* resource is responsible for eye movements.
- The *voice* resource can be used for speaking, e.g. for crew coordination, see section 5.2.1.1.2.
- The *hand* resources (left hand and right hand) can be used to manipulate instruments in the environment.
- The *foot* resources (left hand and right hand) can be used to manipulate special instruments in the environment (Not needed in HUMAN).

The foot resources are currently not needed in HUMAN, and are not yet implemented. The other resources will be described in the following sections.

5.2.1.3.1 Motor: Eye resource

The eye motor moves the gaze, and maintains the current position, as long as there are no requests for a shift to a new instrument. This assumption is made, in order to simplify the model. Without this assumption, a theory would be needed, that describes how long a fixation is maximally kept before people start to look somewhere. In addition, then a theory where to look without top-down control is needed too.

Each eye movement is divided in two phases:

- Phase LLV1: The first phase is a preparation phase, where the head and eye movement to a desired AOI is prepared. This phase is about 140 to 200 ms normally distributed around 170 ms, including the retrieval of the instrument position from the memory. After the position of the instrument is determined, e.g. the position of the PFD, see vector [2] in Figure 5-9, the position of the instrument is transformed into the coordinate system of the eyes, so that the angle α between the actual optical axis of the eye V_1 (e.g. pointing towards

the PFD, see vector [4] in Figure 5-9) and the desired optical axis V_2 (e.g. pointing toward the Navigation Display, ND, see vector [5] in Figure 5-9) can be computed:

$$\alpha(V_1, V_2) = \arccos\left(\frac{V_1 \cdot V_2}{\|V_1\| \cdot \|V_2\|}\right) = \arccos\left(\frac{x_{PFD} \cdot x_{ND} + y_{PFD} \cdot y_{ND} + z_{PFD} \cdot z_{ND}}{\sqrt{x_{PFD}^2 + y_{PFD}^2 + z_{PFD}^2} \cdot \sqrt{x_{ND}^2 + y_{ND}^2 + z_{ND}^2}}\right)$$

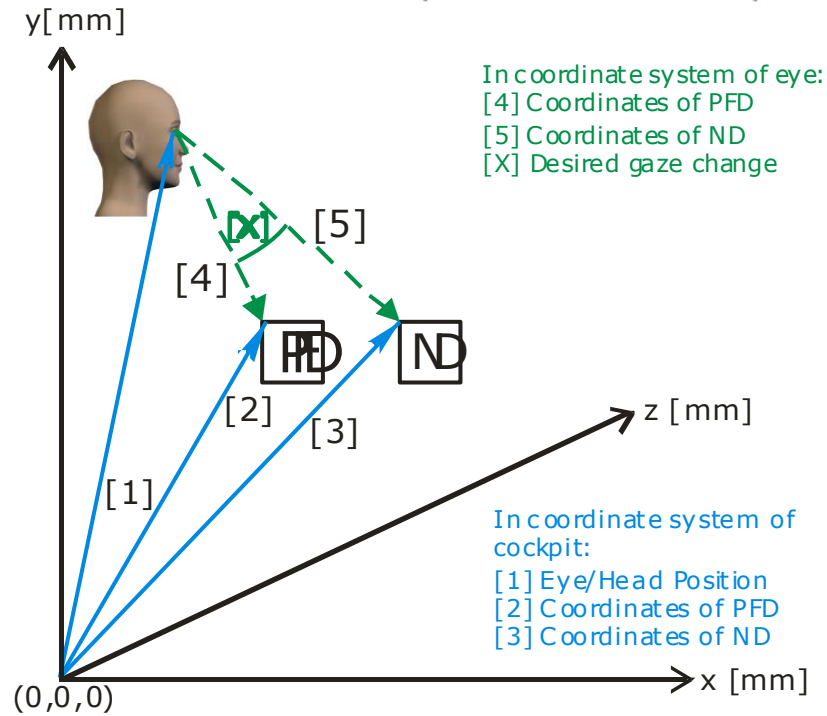


Figure 5-9: Comparison of different coordinate systems

- Phase LLV2: The second phase involves a specific time needed to move head and eyes in order to perform the desired gaze change to the new instrument, e.g. the ND. The model for the combined movement of head and eyes is based on the concepts of (Freedman 2001). The contribution of head and eye, respectively, to the total change α is calculated as follows, approximating the data given in (Freedman 2001):

$$EYE_{contrib}(\alpha) = \begin{cases} \alpha, & \alpha \leq 20 \\ \min(0.31 \cdot \alpha + 13.6, 40), & \alpha > 20 \end{cases}$$

$$HEAD_{contrib}(\alpha) = \alpha - EYE_{contrib}(\alpha)$$

The eye contribution is limited to 40 degrees, due to the physical design of the eyes. The speed of the eye and head are approximated functions from (Freedman, 2001).

5.2.1.3.2 Motor: Voice resource

As described in section 5.2.1.1.2, a speech component is needed for the crew coordination. This has been implemented in the voice motor resource. The Vocal

term in the procedure (Vocal (<spoken text>, <message>, <receiver>)) has already been described in the section on crew coordination. While the <spoken text> reflect the real sentences that are pronounced, the <message> reflects the symbolic meaning of the spoken text. The speech is estimated according to a speech rate of 190 words per minute, which is a mean value for non-native English speakers. It is also possible to add noise to the speech rate, so that a broader range of speech rates can be predicted. The component then starts speaking (symbolically), and stops when the time has passed by. After the time has been passed, the "message" is send to the receiver. The counterpart, the acoustic component is also very simple. As soon as the partner starts to talk, it "listens", and when the speech is finished, the acoustic percept component stores the message, as given in the vocal action, in the memory. This could be for example trigger a reactive rule.

5.2.1.3.3 Motor: Hand resource

The old hand component was very simple. In the procedure it was only possible to use an adjust action (Motor(<instrument>, <new value>)), which was then executed in a fixed time. For the new hand resource we extended the procedure language with more specialised actions, which can also be found in other cognitive architectures, like ACT-R/PM, EPIC or APEX (Anderson et al., 2004; Kieras and Meyer, 1997; Freed, 1998). The following actions have been implemented:

- Move: move the resource to an object (an AOI in the topology)
- Unguided_move: unguided move to the resource (without the eyes).
- Grasp: grasp the object, if resource not already moved to this instrument, move action is automatically done
- Release: release the object (precondition: grasped object before)
- Adjust: adjust the object to a new value (e.g. dial in a value in a potentiometer, shifting the gear, steer the wheel, ...)
- Type: type in a word or value into a keyboard (like) AOI (a grasp of the keyboard has to be executed before)
- Mouse-move: move the mouse to a new location (a grasp on the mouse has to be executed before)
- Mouse-Click: click with a mouse on a location (precondition: mouse grasped)
- Mouse-double-click: click with a mouse twice on a location (precondition: mouse grasped)
- Push: push a physical button
- Pull: a physical object (e.g. altitude selector, direction indicator)
- Move_eye: move the eye to new instrument
- Speak: Speak a text

The timing for these actions has been defined as follows (A=Distance to instrument, W=Width of instrument, H=Height of instrument, D=Depth of instrument):

Move	2D case: $MT = a + b \cdot \log_2 \left(\sqrt{\left(\frac{A}{W}\right)^2 + \mu \left(\frac{A}{H}\right)^2} + 1 \right) + \text{time for eye movement}$ 3D case: $MT = a + b \cdot \log_2 \left(\sqrt{f_W(\theta) \left(\frac{A}{W}\right)^2 + \mu \left(\frac{A}{H}\right)^2 + f_D\left(\frac{A}{D}\right)} + 1 \right) + \text{time for eye movement}$ With $a=55 \text{ ms}$, $b=50$, $\mu = \frac{1}{9.2}$, $f_W(0^\circ) = 0.211$, $f_W(45^\circ) = 0.242$, $f_W(90^\circ) = 0.717$, $f_D(0^\circ) = 0.194$, $f_D(45^\circ) = 0.147$, $f_D(90^\circ) = 0.312$
Unguided_move	2D case: $MT = a + b_1 \cdot \log_2 \left(\sqrt{\left(\frac{A}{W}\right)^2 + \mu \left(\frac{A}{H}\right)^2} + 1 \right)$ 3D case: $MT = a + b_2 \cdot \log_2 \left(\sqrt{f_W(\theta) \left(\frac{A}{W}\right)^2 + \mu \left(\frac{A}{H}\right)^2 + f_D\left(\frac{A}{D}\right)} + 1 \right)$ With $a=55 \text{ ms}$, $b_1=50$, $b_2=500$, $\mu = \frac{1}{9.2}$, $f_W(0^\circ) = 0.211$, $f_W(45^\circ) = 0.242$, $f_W(90^\circ) = 0.717$, $f_D(0^\circ) = 0.194$, $f_D(45^\circ) = 0.147$, $f_D(90^\circ) = 0.312$
Grasp	200 ms + guided movement time
Release	200 ms
Adjust	$\text{timeForFirstUnit} + ((n - 1) * \text{timeToChangeOneUnit})$
Type	100ms per character + grasp of keyboard (100ms/char is from ACT-R/PM)
Mouse-move	$MT = 100 \cdot \log_2 \left(\frac{A}{W} + 1 \right)$
Mouse-Click	250ms*number_of_clicks
Push	1000ms (includes grasp and release)
Pull	1000ms (includes grasp and release)
Move_eye	See section 5.2.1.3.1
Speak	190 words/minute

Remark I: a is the reaction time, and b is a grade of difficulty

Remark II: in order to create some noise, the parameter could be varied, as well as some noise on the fixed values is possible.

Remark III: for the actions release, adjust, type, mouse-move, mouse-click, push, pull the resource has to be grasped before. A grasp may include a movement.

At certain points when the hands are performing an motor action, hand-eye coordination is needed. Before the hand movement is started, the eyes have to fixate the instrument, such that the hand can find the instrument (except an unguided move has been initiated). The eye-fixation is kept until the movement has been performed. When an adjust action is performed, the instrument is fixated until the value is adjusted. If a feedback instrument is given, this instrument is fixated

instead of the instrument where the hand is manipulating. When the action has been performed, the eye is released, and can perform another action.

5.2.1.4 Memory

In CASCaS we do not have separate components for short term memory (STM) and long term memory (LTM), but one memory component, where the distinction between short term memory and long term memory is made on the level of activation a memory item has. Thus, conceptually a distinction between LTM (and long term working memory) and STM is still present. But, as described e.g. by Anderson STM and LTM overlap: "Since working memory contains traces from long-term memory and since new traces in working memory may be permanently encoded in long-term memory, working memory and long-term memory overlap in terms of their contents." (see Anderson 1983). Therefore it seems – on an implementation level – natural to implement STM and LTM in one component, in order to allow an activation based transfer from STM to LTM (Anderson 1983). As one can still distinguish between STM and LTM, based on the level of activation, this shouldn't be contradictory to studies which show that these memories exist. We do not say that there are no differences between declarative memory, procedural memory, episodic memory, etc, we only implement this in the same structure, to allow easier learning and also a tighter integration of these concept.

The memory component consists of declarative and procedural knowledge. These kinds of knowledge are treated in distinct ways. It is combined with similar structures in one component to allow building up associations between declarative and procedural knowledge.

This section will at first describe the declarative knowledge and the processes involved in changing and using it. Afterwards the procedural knowledge is described and its relation to the declarative knowledge.

5.2.1.4.1 Declarative Knowledge

The declarative knowledge is information that a human can use when doing a particular task. It builds up a mental representation of the human's environment. Airport staff members or pilots for example will normally have some knowledge about airports in different cities and their airport codes.

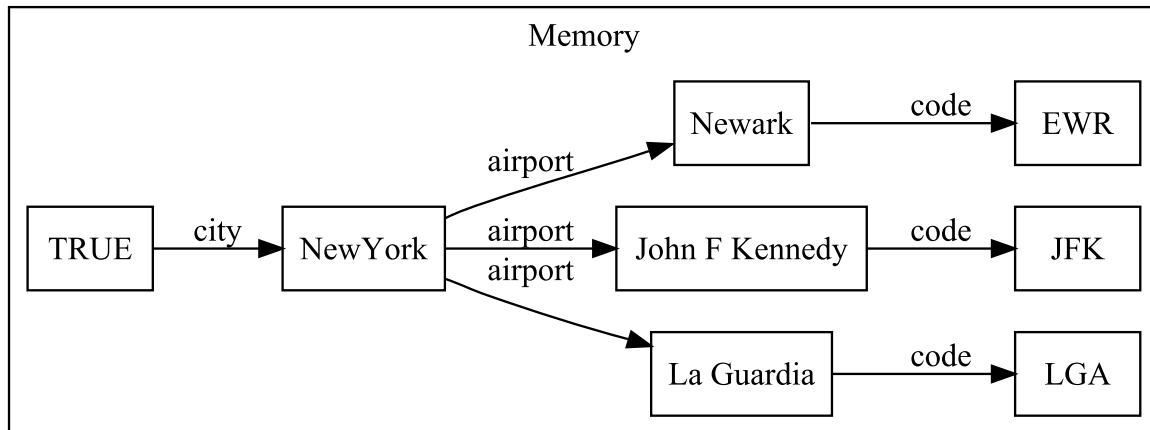


Figure 5-10: Representation of knowledge about airports

A possible representation of an extract of this knowledge might look like the structure in Figure 5-10. This is the way the declarative knowledge is organized in CASCAS. It is constructed as a semantic net with items and labelled associations between the items (for semantic net, see Sowa 1987, or Shapiro 1987). The items in the net represent a piece of information that stands for any kind of concept the human is aware of. Each item i has a value v_i . These can be literal values like "New York" and "JFK" or real and integer numbers. The concept an item stands for and the value of the item can be the same, like the letters "JFK" are really the airport code, whereas the letters "New York" are just the name of the city. For simplicity it is normally sufficient to use the name of concepts as their representation. But the memory component also allows to explicitly distinguishing between these.

For this reason it is also possible to have items without a value. So when distinguishing the concept of the cities and airports in Figure 5-10 from their name a mental model representation can look like the one in Figure 5-11. There are now items without a value ($v_i = \emptyset$) that represent abstract concepts, which normally have some attributes associated with it.

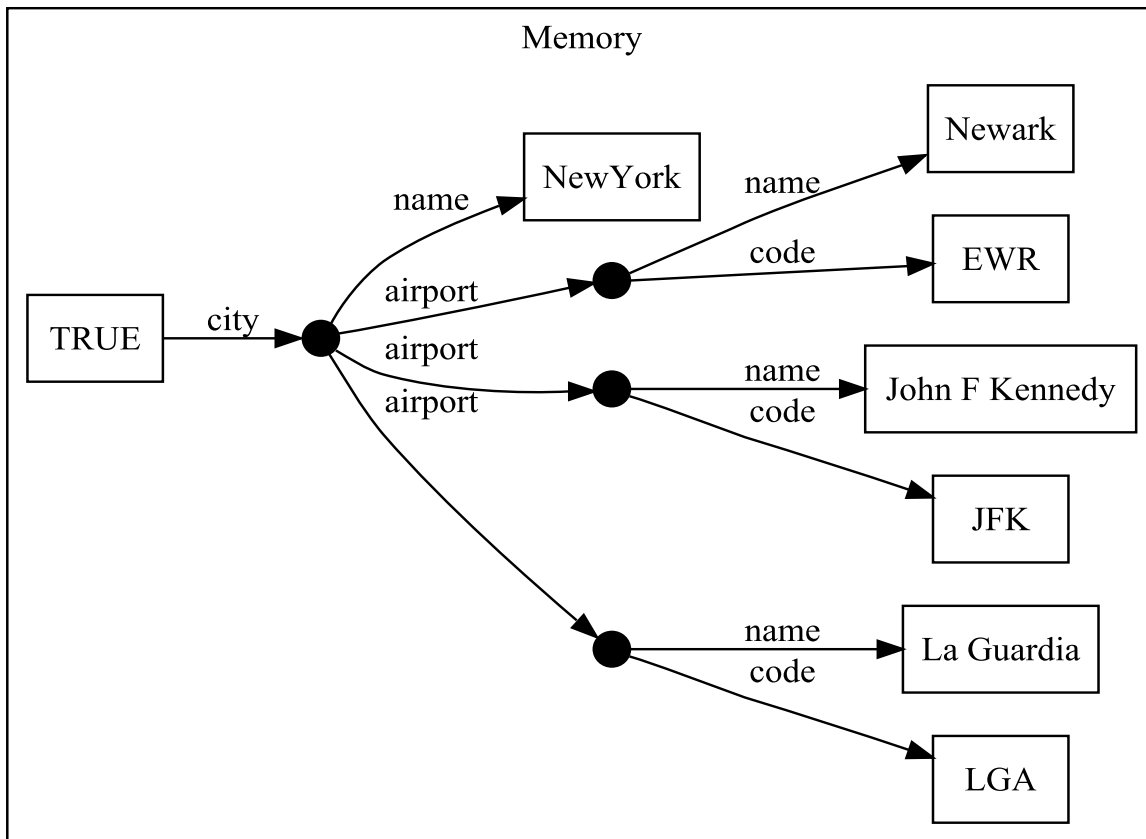


Figure 5-11: Mental representation with nodes having no value

These different representations already indicate that for the same things there can be different mental model representations. There is no complete model because humans have a huge amount of knowledge which differs with each individual. Therefore there is also no model that is correct for every person. Especially it is not possible to verify a mental model, because these structures are only an abstraction which cannot be derived by inspecting the human brain.

To use the model within CASCAS it is possible to specify a pattern of interest, which is then searched in memory. This is for example done when the architecture searches for rules to be fired. Every left hand side of a rule does specify such a pattern. If for example a model with a mental model like in Figure 5-11 is asked to tell the name of the city, where the airport with name La Guardia and code LGA is, it could use a rule like the rule depicted in Figure 5-12.

```

id=1, name=response_city, type=regular
Retrieve, city.name
Condition, city.airport.name == 'La Guardia'
Condition, city.airport.code == 'LGA'
-->
Vocal, 'The city is #city.name#'
;

```

Figure 5-12: Rule 1 to retrieve the city where the airport La Guardia is located

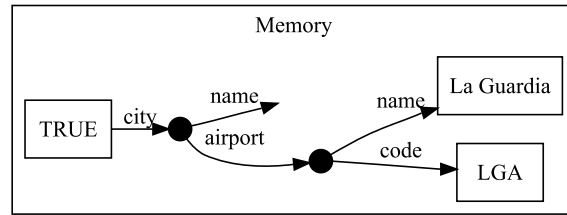


Figure 5-13: Pattern for airport search in Rule 1

To check whether the rule can be fired, the pattern shown in Figure 5-13 will be searched in memory. If there is a part in the memory matching this pattern it will be retrieved and can then be used to execute the actions on the right hand side of the rule.

5.2.1.4.1.1 Activation

The retrieval of a matched pattern is not only dependent on the existence of the pattern in memory; it also depends on the activations of the involved memory items.

Each item in memory has an activation value that changes over time. The activation A of item i at time t is calculated with:

$$A(t, i) = \beta - \text{decay} * \ln(t - t_{w_i}) + S(t, i) \quad (1)$$

With t_{w_i} being the last time that the item i has been written to memory; decay being a user defined value, specifying how fast the activation reduces over time and $S(t, i)$ being a value ≥ 0 , which will be described later.

The first addend in equation 1 is a simplification of the term for base level activation introduced in Anderson 1998 which is based on work of Anderson and Schooler 1991. The here used simplification eases the computation of the activation during simulation. But it also introduces a new parameter β for which the model has to choose a value. Some passages below it will be explained, how β is used in CASCAS. In Figure 5-14 you can exemplarily see how the activation for an item decreases over time after it has been written to memory.

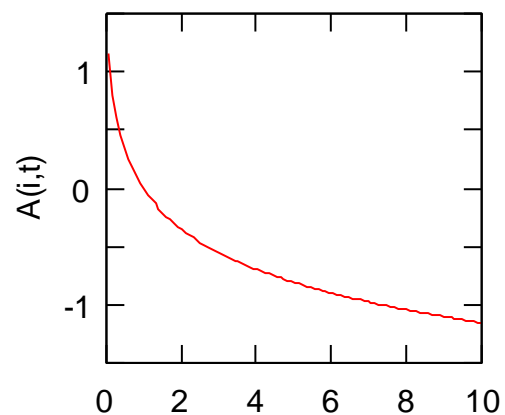


Figure 5-14: Activation decrease with $\beta = 0$, $\text{decay} = 0.5$ and $t_i = 0$

Another source of activation for an item is based on the concept of spreading activation, see Anderson 1983. It is expressed in the above equation by the $S(t, i)$ summand and will be explained after the procedural knowledge has been described.

5.2.1.4.1.2 Retrieval of items

The activation is used to determine which items in memory can currently be accessed by the model. An item can only be used, if its activation is above a certain threshold. So at any time only a part of the mental model can be retrieved by the model. If an item i is accessible, the time RT it takes to retrieve it is calculated by:

$$RT(i) = lf * e^{A_i} \quad (2)$$

with lf being the latency factor – a constant model wide factor. This equation again is from the work of Anderson and Lebiere (Anderson & Lebiere, 1998). Even though the activation process relies heavily on the concept that is used inside the ACT-R architecture (for detailed information see the webpage for the current ACT-R version 6, <http://act-r.psy.cmu.edu/actr6/>), CASCAS has a slightly different process of retrieval. In ACT-R chunks from declarative memory are retrieved into the retrieval buffer, when a fired rule requests such retrieval. After one chunk has been retrieved a rule will be fired, that matches the content of this (and all other) buffer. In CASCAS these steps are done simultaneously. The patterns of the left hand side of all rules of the current goal are matched against the current memory state. One of the matching rules will be selected. This mechanism is more like the rule matching mechanism in SOAR, where the rules are also matched against the whole memory content. And also in SOAR there have recently been effort to integrate Anderson and Lebiere's subsymbolic mechanism of base level activation, see (Laird 2008, Nuxoll & Laird 2004), but there it is not used to determine whether a working memory element is accessible or not.

Equation 2 is used in ACT-R to calculate the time from requesting a chunk into the retrieval buffer to the time it is actually placed in the buffer. Because in CASCAS this step is done implicitly when selecting a rule, the retrieval time equation is used in CASCAS to calculate the time it takes to select a rule.

5.2.1.4.1.3 Item evaluation

The sections above showed a slightly simplified version of memory items. Looking at the mental model in Figure 5-10, it is obvious that this model won't be changed very often, because it is very improbable that a city will change its name or that an airport will change its airport code. But many other things do change very often.

This is normally the case for information, which the model receives from percept component. All information that enters the architecture via the percept component will be stored as declarative items in memory. As the environment changes, the corresponding items in the memory will also change.

But these changes should not overwrite the old values, because humans do not just remember the things they currently see. They can also remember (at least for some time) what they have seen before. To take this into account, we do not store the values directly in the items as shown in Figure 5-10. Every time a new value for an item i of the mental model is written to the memory a new item e will be created and linked to the item i with a special evaluation association. For every new value a new item will be created. The t_{w_i} value (see equation 1) for this item will be set to the current time to activate it. If a value has been written previously, the t_{w_i} value of the existing item will be set to reactivate it. Figure 5-15 will demonstrate this. It shows the mental representation of a flight mode display. For every evaluation item the activation over time is also shown. The first time t_1 the model looks at the display and perceives a value for it (here _BLANC_) items will be created like in figure a). When the value ALTS appears at time t_2 on the display, a new evaluation item will be created like in figure b). When the value of an existing item is perceived the item is reactivated like the evaluation item _BLANC_ in figure c).

The value v_i for every item i that stores no value directly, is now taken from the evaluation item with the highest activation that is associated to i .

$$v_i = \begin{cases} \emptyset, & \text{if } E_i = \emptyset \\ v_j \mid A_j = \max(v_k \mid k \in E_i), & \text{else,} \end{cases} \quad (3)$$

with E_i being the set of all items that can be accessed from i via an evaluation association.

The use of this construct will be shown in section 5.2.1.4.2.1 where spreading activation is described.

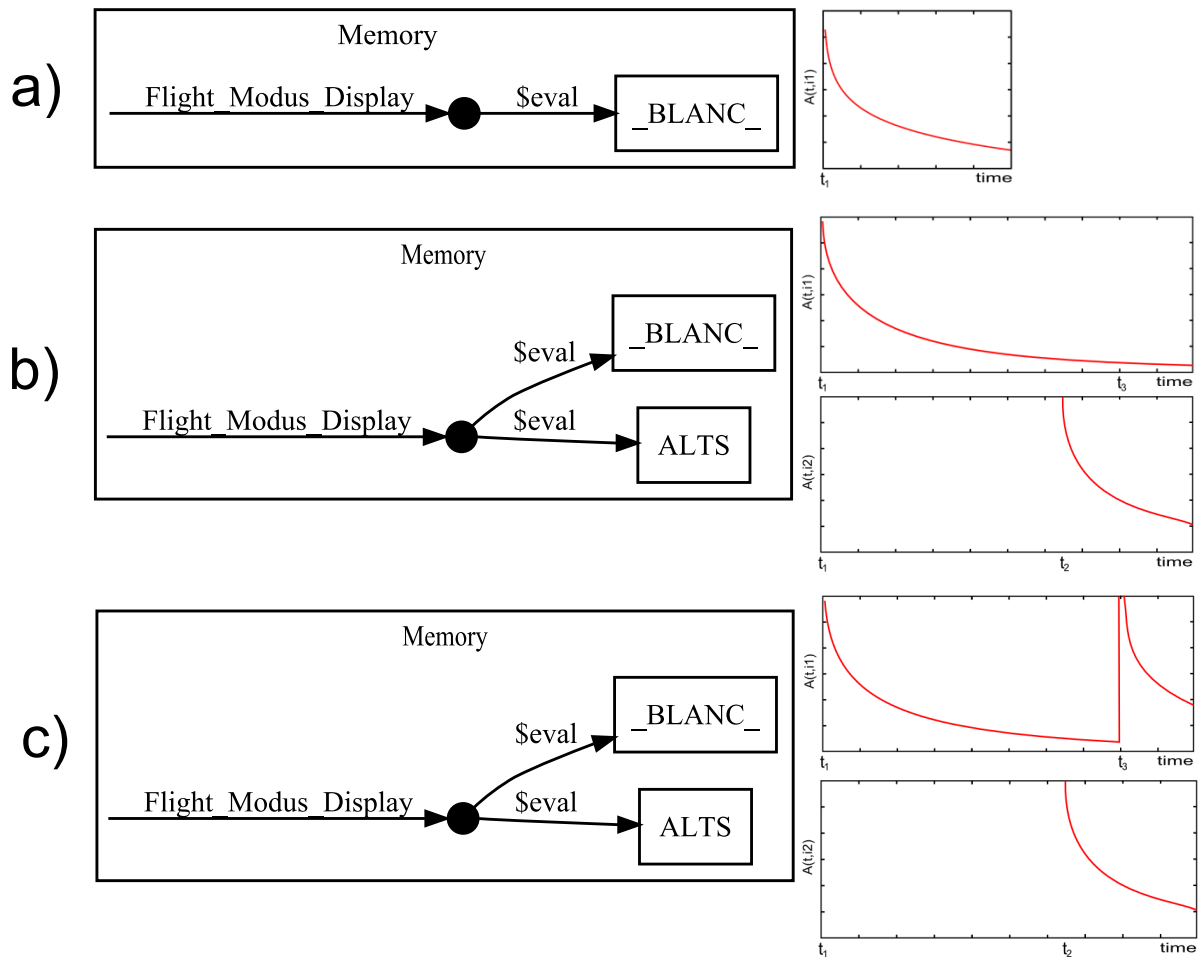


Figure 5-15: Change of mental model for Flight_Modus_Display. a) Perceiving _BLANC_ at t_1 . b) Perceiving ALTS at t_2 . c) Perceiving _BLANC_ at t_3 .

5.2.1.4.2 Procedural Knowledge

The procedural knowledge describes how the model can use its declarative knowledge to solve a certain task. It is represented by rules, see section 4.4.3.

A rule is represented in memory as one item that has associations to each element on its left and right hand side. As each rule is only applicable when a certain goal is active, each rule has also an association to an item in memory that represents this goal. An fictive example of a task for an driver to keep distance to the car in front can be seen in Figure 5-16. The goal "keep_distance" has two rules and each of these rules again has associated items for its left and right hand side elements.

Reactive rules as a special case do not have goals and are therefore not related to any goal (see Rule 3) in Figure 5-16.

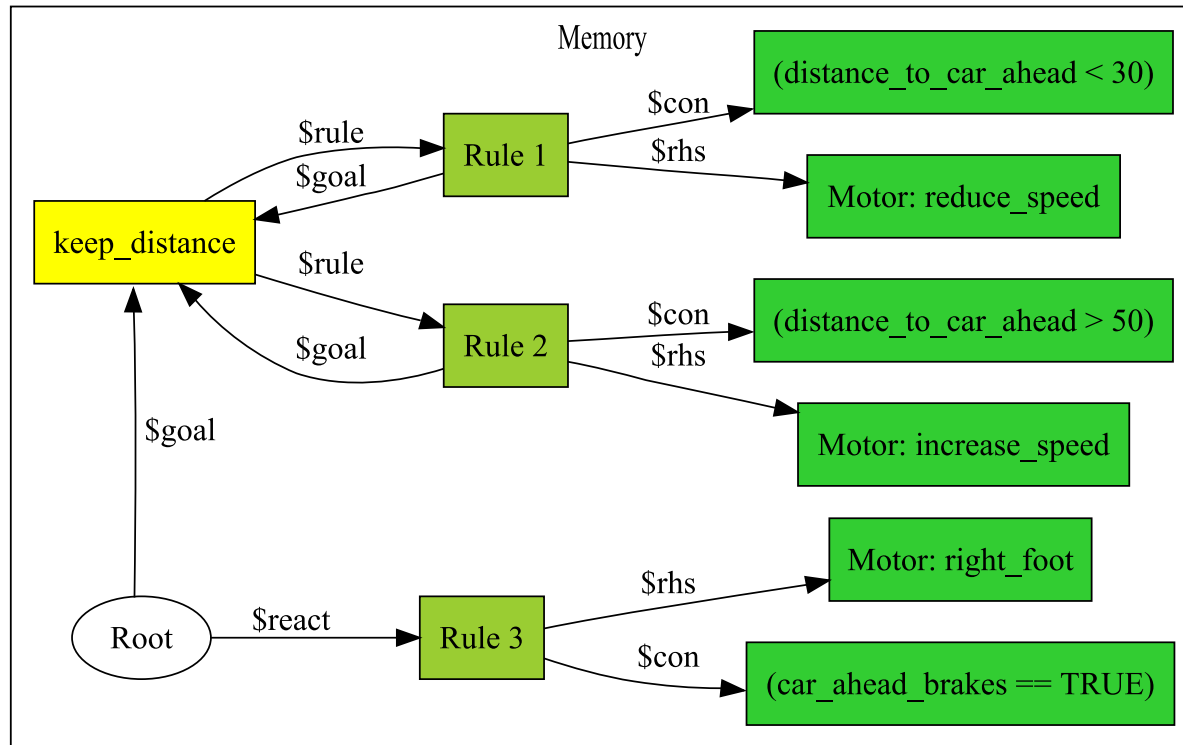


Figure 5-16: Goals and Rules in procedural memory. The yellow item represents the goal. The light green items represent rules. And the green items are the left and right hand side elements of rules.

In the current version of CASCAS the representation of rules and their attributes as items in memory has no specific reason but to store the elements in the architecture. This might be changed in later versions. But what is of particular interest is the representation of goals as items in memory.

Goals play a central role on the architecture, because they represent the intention of the human model and are the main guide for the selection of rules on the associative layer.

5.2.1.4.2.1 Spreading activation

As short term memory (STM) we denote items in the memory that are never or very seldom written to memory after their first creation. These items are hard to remember for the model, because their activation is constantly decreasing. They are only accessible some short time after they have been written to memory.

But there is also long term memory (LTM) which last longer in memory and which will be activated when the human (model) needs it, without updating the items from perceptual information. Take for example the PIN code of your bank card. You immediately know it when standing in front of the bank cash machine, but you aren't aware of it during driving or flying; even so you can remember it. But

normally remembering will move your attention away from the task of driving or flying.

To model long term knowledge there is an architectural mechanism that automatically creates associations between items of declarative knowledge and goal items. Every time an item is created in memory a contextual association between this item and the current goal is created. This expresses that this item seems to be relevant in the current context, i.e. the current goal. If the item that shall be written already exists, it will be reactivated (see section 5.2.1.4.1). If in this case there is also already a contextual association to the current goal, the strength of this association will be increased by a constant factor λ - the association learn rate.

As a result every goal item can have many associations to declarative items. These are the items that have somehow been used during the time this goal was active. The relative strength between two associations originating from the same goal reflects the relative frequency of occurrence of the associated items during the time this goal was active.

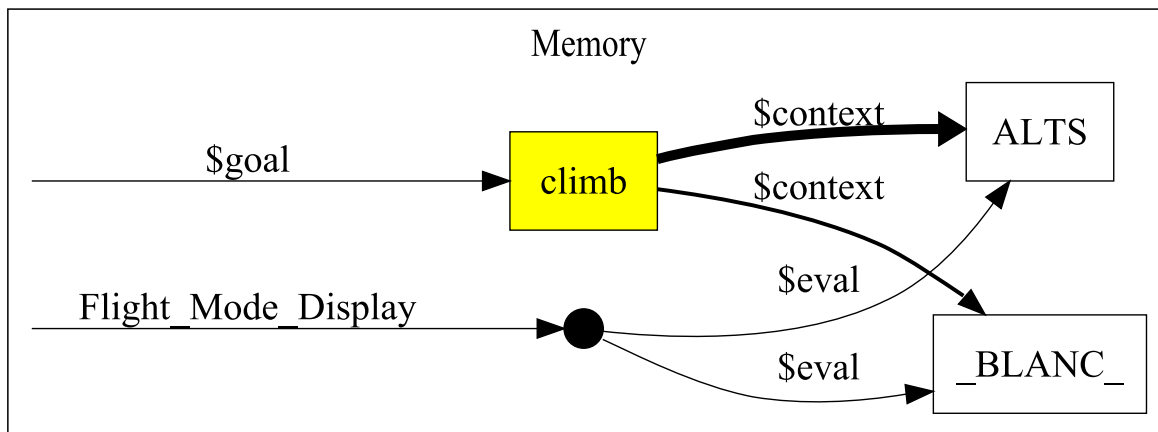


Figure 5-17: Contextual associations to different evaluation items

Take for example the item for the flight mode display in Figure 5-15. It has two evaluation items. When the flight mode display is watched during the goal climb, the percept component will write the values to memory. In this process contextual associations from the goal to the evaluation items will be created respectively be strengthened. This can be seen in Figure 5-17.

In the current version the process described above is the only process that creates contextual associations. This might be changed in later versions.

Let the association from item i to item k be

$$r_{i,k},$$

the strength of association r be

$$\text{str}(r),$$

the set of all goal items be

$$G,$$

the number of associations from any goal to item i be

$$n_i^{\text{in}} = |\{r_{g,i} \mid g \in G\}|.$$

With these parameters, the described construct of contextual associations is used to calculate the $S(t, i)$ factor in equation 1.

$$S(t, i) = \frac{1}{n_i} * \sum_{k=1}^{|G|} (str(r_{g,i}) * A_{g_i}) \quad (4)$$

This just means, that the activation of each goal is distributed along the associations to all associated items. Currently this is only done for contextual associations. As these associations currently are only originating from goal items, the spreading of activation will start at the goal items and immediately stop after one step.

Here the goals serve as sources of activation. At each time only the currently active goal in the goal module has an activation value unequal to zero. Every time a new goal is selected the goal item receives a predefined activation value and the activation of the previously selected goal is set back to zero. In practice this saves us the summation over all goals, while just using the active one.

Equation 4 has some severe theoretical problems. Because the strength of an association can theoretically rise unbounded, the activation an item gets via the $S(t, i)$ parameter can also rise unbounded. In future versions the influence of the association strength will probably be related to the strength of neighbouring associations to prevent unbounded growth. For practical reasons the equation is so far left as is, because spreading activation has until now only be used to show the effect of learned carelessness. For this phenomenon the simply equation used here is sufficient. But in future versions it will probably be changed.

It has been shown how LTM and STM are accessed in the architecture. What makes up working memory in CASCAS is the portion of STM and LTM which has a high enough activation, to be retrievable. And even in this subset of memory there are items that are easier to access then others due to higher activation. This definition of working memory is very similar to the one in (Anderson 1983). As a consequence the working memory definition of CASCAS has no clear boundary.

5.2.2 Associative Layer

5.2.2.1 Reactive Behaviour

In section 4.4.3 the rule based language has been described. The rule based language foresees to have only rules linked to one goal. In contrast to this, reactive behaviour is always active, i.e. reactive behaviour is independent from a goal, but it may trigger a new goal that becomes active. Reactive behaviour can be used e.g. for modelling crew coordination – if the crew member ask for something, the other has to *react* on this. But also for certain events in the environment reactive behaviour is needed, e.g. when warning lights or warning sounds are present. In order to model this, the concept of *reactive rules* has been added to the procedure language. Reactive Rules have the same structure and semantic like regular rules, with the exception that they do not belong to a goal. So, while regular rules have

also a goal on the left hand side (as part of the condition, that this goal must be active in order to fire this rule), the reactive rules have only a condition on the LHS. As soon as the condition of a reactive rule evaluates to true, the rule is fired, independently from the current active goals.

5.2.2.2 Multitasking

While Pilots perform a certain task, e.g. re-plan because of bad weather, communicate with ATC and so on, they are also required at the same time to monitor the flight. Also the standard procedures foresee to monitor different instruments at one time, in order to trigger certain motor actions, for example in a Piper Chayenne, after the takeoff, the pilot has to monitor the airspeed, in order to retract the flaps when needed. In parallel, he is required to monitor the vertical speed and to stabilise this speed at the desired rate. Then, he has to monitor the modes and the altitude, in order to check the correct level-off of the aircraft.

All these tasks have to be done in parallel. Additionally, external communications can occur, which requires task interruptions. As the predicted behaviour from the cognitive model should be as realistic as possible, multi-tasking has to be considered in the architecture.

5.2.2.2.1 Existing Approaches

Most cognitive architectures do not support multi-tasking from scratch, but have been extended in this direction, e.g. ACT-R has been extended by Salvucci (Salvucci, 2005) for his modelling efforts of driver behaviour. MIDAS, an agent-based architecture from NASA, uses Shankars' "Z-Scheduler" algorithm (Shankar 1991). MIDAS version 3.0 allows the specification of procedural knowledge in Freed's language PDL from APEX (Freed 1998). This allows the integration of active procedures, which are similar to the reactive capabilities we implemented in our architecture. In contrast to this, EPIC (Kieras & Meyer 1997) allows by default massive multitasking capabilities within the rule interpreter, but to serialize actions specific multitasking rules need to be specified, which realize different concepts for interleaving of compound continuous tasks.

Within CASCaS, multitasking capabilities will have different characteristics:

1. On the associative layer, several goals need to be interleaved: e.g. "monitor airspeed" and "monitor vertical speed". These two tasks will be interleaved rule-based on the associative layer, i.e. the modeller of the procedure has to care on the interleaving in the procedure itself.
2. The reactive capabilities which are described in the previous section allow a further way to interrupt an ongoing goal, by adding a new high priority goal onto the agenda. Therefore this is also an aspect of the multitasking capability of the associative layer. Such a reactive goal may be used to handle perceived events based on situation specific characteristics, which means that the event may be critical in one situation but not in another. For

this purpose different set of rules may be integrated in the procedure which comprises situation assessment.

3. There will be no Multitasking on the cognitive layer.
4. The cognitive Layer and the associative layer work independent, and in parallel, also there are points of synchronisation, e.g. when the associative layer is in a impasse and waits for the cognitive layer to solve the problem.

Regarding the associative layer, the model will include a goal queue, based on the general executive for multitasking of Salvucci. It will allow the interleaved execution of continuous goals. The goal queue of the associative layer will interact with the goal agenda and goals will have different states of execution:

1. Initially, each goal which is put on the agenda (start goal "acquire_task" or later on new goals added during "fire_rule") is in the state blocked.
2. During the goal selection phase each goal is checked by the goal agenda for preconditions. If no precondition exists, the goal state changes to "selectable".
3. Each selectable goal on the agenda is put on the multitasking agenda and then set to "active".
4. The active goals are scheduled as long as a specified rule terminates the goal (no more subgoals or by specific interrupt / termination event). Now the goal is queued out and its state on the agenda is set to "done".
5. Goals in the state "done" are removed from the agenda. Such a goal is furthermore removed from the precondition list of those goals, which are in state "blocked".

A goal which is put on the agenda (normally during "fire_rule") can be associated with a certain deadline. After expiration, the goal is scheduled with priority. This mechanism can be considered as a "mental clock" and is useful for example in the following situation:

- A continuous goal might be to schedule a monitoring task which terminates if the monitored instrument shows a specific value. At the beginning of the task it may take a while until the target value is reached, therefore the pilot decides to suspend the goal for a certain amount of time. Looking at the instrument later on (recovery of the goal), the target value may be quite near and therefore he schedules the task much more often. This may be modelled with a set of rules, which make situation dependent assumptions about criticality and the chronological characteristic of the value.

The mental clock and the recovery of a suspended goal can be a possible source of error, if the activation of the goal pattern disintegrates due to different activation patterns in the working as well as in the procedural memory. For more complex tasks, which do not only rely on the simple perception of a single value but on a more complex set of information these may be outdated or simply forgotten and the

task cannot be recovered at the point where expected. In this case additional perception and information integration is needed to get up to date.

With the help of continuous goals it is also possible to model interleaving in the rules. Considering the following example of tasks (modelled in AMBOSS):

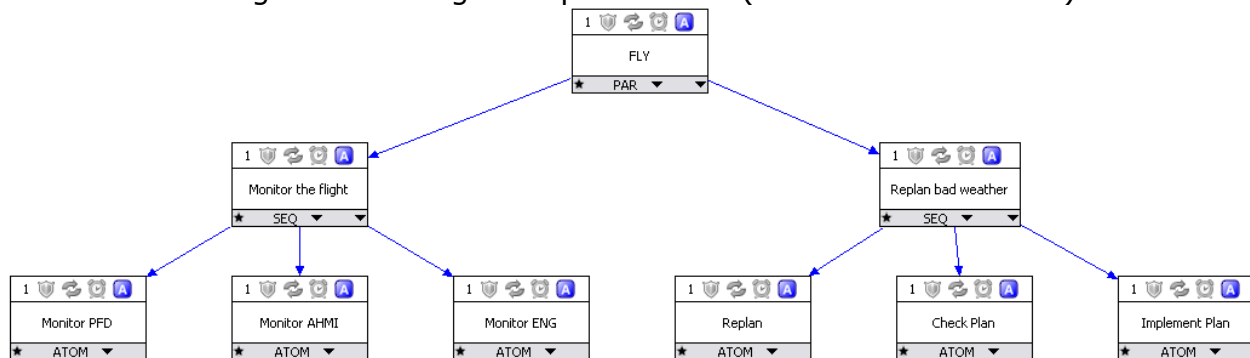


Figure 5-18: Example for Multitasking

The top goal is “FLY”, and “Monitor the flight” is always active. Now we have the additional goal “Replan bad weather”. The monitor task and the replan task have several sub-goals. Normally these sub-goals are again divided into several sub-goals, but for this example it’s assumed that the sub-goals of monitor and replan are atomic. Without interleaving and continuous goals, CASCaS would perform all goals once, with the following sequence of active goals: FLY, Monitor the flight, Monitor PFD, Monitor AHMI, Monitor ENG, {FLY, until reactive rule starts Replan bad weather}, Replan bad weather, Replan, Check Plan, Implement Plan. If we make monitor the flight continuous, then we would get for example the following sequence of active goals: FLY, Monitor the flight, Monitor PFD, Monitor AHMI, Monitor ENG, {(Monitor the flight, Monitor PFD, Monitor AHMI, Monitor ENG) until reactive rule starts Replan bad weather}, Replan bad weather, Replan, Check Plan, Implement Plan, Monitor the flight, Monitor PFD, Monitor AHMI, Monitor ENG, {(Monitor the flight, Monitor PFD, Monitor AHMI, Monitor ENG) until scenario end}. Also better, even this is not the desired behaviour. Ideally, we would see interleaving between the goals. This is especially needed, if a task has a long duration, e.g. the “Implement Plan” goal. During this, it is necessary to monitor the flight too. In order to model this, we build this interleaving into the rules. Let’s assume we translated the previous model into the following rules:

```

Rule 1
    Goal (FLY)
→
    Goal (Monitor the flight, mode=continuous)
;

Rule 2
    Goal (Monitor the flight)

```

→

- (1) Goal(Monitor PFD)
- (2) Goal(Monitor AHMI, precondition=1)
- (3) Goal(Monitor ENG, precondition=2)

;

For modelling the interleaving, we use a kind of state-automaton that we build into the rules:

Rule 1

Goal(FLY)

→

MemWrite(state, PFD)
 Goal(Monitor the flight, mode=continuous)

;

Rule 2

Goal(Monitor the flight)
 Condition(state==PFD)

→

Goal(Monitor PFD)
 MemWrite(state, AHMI)

;

Rule 3

Goal(Monitor the flight)
 Condition(state==AHMI)

→

Goal(Monitor AHMI)
 MemWrite(state, ENG)

;

Rule 4

Goal(Monitor the flight)
 Condition(state==ENG)

→

Goal(Monitor ENG)
 MemWrite(state, PFD)

;

Because of the goal agenda and the semantic of continuous goals (automatically re-appended at the end of goal-agenda), we get the desired interleaving. At the first selection of "FLY", the state variable is set to PFD. Then, in the rule selection for the goal "Monitor the flight", only the condition of rule 2 is true, and can be fired. After the goal "Monitor PFD" is finished, the state variable is changed to AHMI. But as the rule has been fired, the goal "Monitor the flight" is done. Because it is a continuous goal, it is appended to the goal queue again. If there is another goal, like "Replan

bad weather”, this goal is now on the first position of the goal queue, and will be selected next. When this goal is finished, again the “Monitor the flight” goal is active, and Rule 3 can be fired, and so on.

5.2.1 Cognitive Layer

In this section, the cognitive layer is described in detail. An overview of the different components in the cognitive layer is given in Figure 5-19.

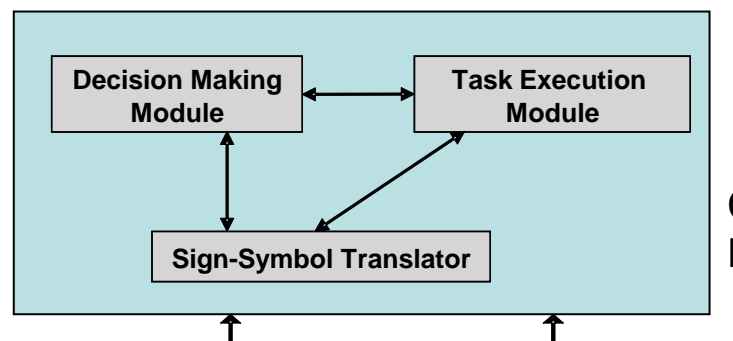


Figure 5-19 Overview of the components in the cognitive layer

In the cognitive layer, we differentiate between the following modules: the decision-making module (DMM), the task execution module (TEM), and the sign-symbol translator (SST).

The decision-making module is also called goal management, and determines which goal is executed. To do this, it needs access to the available signs and symbols, which are stored in the working memory.

The sign-symbol translator (SST) communicates with the Working Memory (WM) in both directions. Signs are sent from WM to SST to enable translation from signs to symbols. Symbols are sent from SST to WM to store these newly derived symbols for future use by the CL. Goals can be selected not only on the basis of signs, but also on the basis of symbols. The DMM and the TEM can send queries to the SST, after which the SST tries to derive the answers, which it stores in the WM.

When a goal is chosen to be executed by the Decision Making Module, the Task Execution Module executes the goal. Tasks (or sub-goals) that the associative layer

can handle are passed to the associative layer. Tasks that involve SST (as they involve cognitive reasoning) are passed to the SST module. Some tasks, e.g. replanning the trajectory, are executed in the task execution module itself.

Figure 5-20 gives an overview of the interaction between the DMM, TEM and SST.

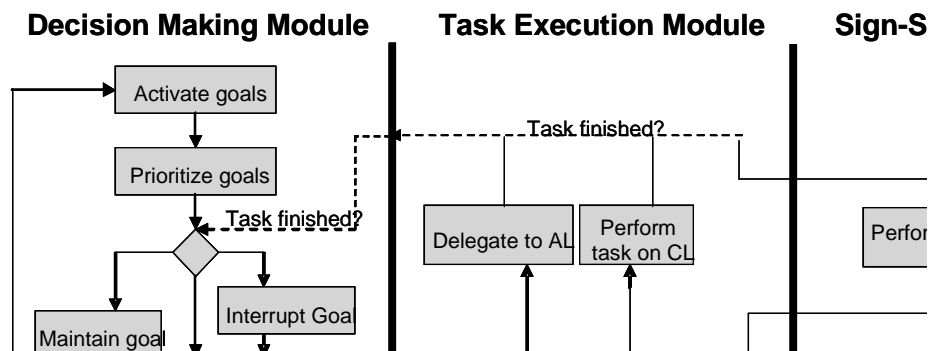


Figure 5-20 Detailed view on the components in the cognitive layer

On the left, the flow of processes of the decision making module is shown. The task execution module runs in parallel to this process. Performing the translation of signs into symbols or symbols into symbols can be part of a task execution. The following sections describe these components in more detail.

5.2.1.1 Decision Making Module

As described in 5.2.2, in the associative layer, behaviour is produced through rules that are associated with goals. Also in the cognitive layer, goals play an important role in producing behaviour. In the following, it is described in detail how goals are organized in the cognitive layer, which role they play in producing behaviour and how they are connected to the goals on the associative layer.

5.2.1.1.1 Requirements of the goals in the cognitive layer

In this section, the requirements for the goals in the cognitive layer are described. The following requirements have been identified:

- the goal organization should be built as a hierarchical task/action model (it should be a tree, that is with no cycles in the path). This is important as the model should include both CL and AL; the procedures implemented in the AL are a continuation of the goal hierarchy of the CL and thus are connected to the goals in the CL. That means that goals of the CL must activate goals in

the AL. There may also be goals in the AL for which no standard procedure can be found, in which case the goal is executed on the CL.

- Every (sub)goal should be formalized in a way that makes it possible to
 - o decide the priority of a goal in comparison to all other goals. This can be just a priority value or a relational function, which determines a (partial) order (as in Soar (Laird, Newell, & Rosenbloom 1987))
 - o decide whether a goal is active. A goal is active if it is applicable in the situation. This means, for each goal, the conditions that make a goal active have to be specified.
- it must be possible to represent the different levels within the goal hierarchy: there are goals that are a task in itself and for which there are conditions that have to be true before it is chosen, and there are (sub)goals that belong to the execution of a higher-level goal, which only condition is that the higher-level goal has to be chosen.
- The goal hierarchy should be set up in a generic way. The cognitive layer should be able to import the normative behaviour automatically, if it adheres to the requirements and specifications that are going to be identified in T2.4 . The normative behaviour can be seen as 'rules of good airmanship', which means non-proceduralized behaviour which is good practice.

5.2.1.1.2 Basic Idea

A suitable method which meets the requirements described above is a hierarchical task network (Russel, Norvig, 2002). We differentiate between the following terms:

- *Active goals*: goals of which the conditions are true and which can be executed (but are not necessarily chosen to be executed). There can be several active goals at the same time.
- *Selected goal*: the goal that is selected (from the set of active goals) to be achieved at a particular point in time. A selected goal thus is always active. Which goal is selected depends on its priority in the particular situation.
- *Top-level goals*: top-level goals are either permanent goals and thus always active (e.g. fly safely, monitor the associative layer) or are reactive goals that need conditions to become active (e.g. replan, emergency descent). These conditions depend not on the activation of a different goal, but on other conditions that are derived from the environment.
- *Lower-level goals (tasks)*: these goals are active when a higher-level goal becomes active. They thus can be seen as goals that have only one condition, namely the selection of the goal they belong to. It can be seen as a goal that implements the achievement of a higher-level goal. In a way, this is what is modelled at this moment in AMBOSS as task models (see 4.4.1). All lower-level goals that belong to a particular top-level goal can be seen as the plan on how to achieve the top-level goal.

As already stated in D1.3, we believe that the goals in the associative layer are an extension of the goals in the cognitive layer. In the associative layer, if a goal is active, it will be executed. If several goals are active, they are placed in a queue and, as described in the multi-tasking capability description (see 5.2.2.2), are all executed. This is not the case for the goals in the cognitive layer; we decided not to implement multi-tasking for the cognitive layer, but to decide which goal deserves attention. Attention then is allocated to a particular (high-level) goal, dependent on the goal decision mechanism.

As mentioned above, we differentiate between two different kinds of goals, namely high-level and lower-level goals.

5.2.1.1.2.1 High-level goals

The following properties of high-level goals can be identified:

- High-level goals have conditions that make them active or not active. These conditions can have as variables either signs or symbols.
- They have a goal name.
- High-level goals have a priority, which depends on
 - o a value set by the domain expert
 - The begin value of the priority can be seen as setting an order on all goals, and represents the possible orders of the normative behaviour. As the decision-mechanism that determines which goal receives attention takes the priorities into account, the begin value should be an approximation of importance of a particular goal in relation to the other goals.
 - o attentional capture. This will be described in detail in 5.2.1.5.
- High-level goals are linked to a corresponding plan. The corresponding plan is what we call the lower-level goals that implement this high-level goal. A lower-level goal can again have a corresponding plan, thus be complex.

Several types of high-level goals can be identified:

- always active goals (e.g. goal to monitor the AL)
- standard goals.
- emergency goals.

These goals can be seen as three different classes of goals; it is possible to give a general ordering of priorities to these types of goals: a monitoring goal has always a lower (original) priority than a standard goal, which always has a lower (original) priority than an emergency goal. It is however possible that this ordering is not always true, especially if one or several biases are applied to the goals.

5.2.1.1.2.2 Lower-Level Goals

Lower-level goals are seen as implementations of high-level goals. They can be interpreted as abstract actions that have to be executed to achieve a goal. Lower-level goals can be complex, thus be implemented by other lower-level goals. The

plan (all lower-level goals that implement the high-level goal) can be a partial order plan or have a strict order.

5.2.1.1.2.3 Goal Management

If, while executing the lower-level goals of a high-level goal, a new high-level goal becomes active, the priorities of these two higher-level goals are compared: if the priority of the new goal is (clearly) higher than the priority of the goal that is being executed, the execution of the latter is stopped and the new goal is attended to. The associative layer does not get a respond on the previous goal so this is set in the multitasking list of the associative layer. This appears to be the most realistic way of reacting: only if another task has a considerable higher priority, a task is interrupted. If a task is interrupted and another task is executed, usually, after having finished the task, it is again evaluated whether the first task is still necessary. The goal has to be evaluated (just as all other goals) to see whether it is still active and should be selected to be achieved. A goal on the cognitive layer that has been interrupted is thus not automatically resumed after the other goal has been reached. It is assumed that on the cognitive layer, executing a goal leads to a change in the environment and in the mental model so that the situation in which the decision to execute the goal has been made has changed. A new evaluation of the situation is necessary to determine whether the goal still has a high priority and should be chosen to be executed again.

The priority of goals are not static, but change over time. For every goal that is active, the priorities are increased over time. A task that has standard a low priority, such as monitoring, increases thus in importance if it is not selected in a long time. Implicitly, temporal deadlines are thus modelled.

5.2.1.2 Task Execution Module

The task execution module determines how a task is to be executed, and runs in parallel to the Decision Making Module. In general, there are three components that can compose a task on the CL:

1. Action to be performed by the associative layer
2. Activation of the sign symbol translator
3. Execution of cognitive steps by the Task Execution Module itself.

For tasks that are executed by the associative layer, a message is sent to the AL. Examples of these tasks are implementing new trajectories into the AHMI, i.e. tasks for which standard procedures are available. Examples of tasks which are performed by the SST are evaluating plans, i.e. cognitive reasoning to assess whether a plan is favourable, or unfavourable. An example of a task which is performed by the TEM itself is the cognitive replanning task, i.e. replanning with a non-standardized procedure. Such a task cannot be executed by the AL and neither by the SST.

5.2.1.3 Sign-Symbol Translator

In this section, the capability of translating signs into symbols is described. As described in D1.3, as a theoretical foundation for the CA, the SRK-Model from Rasmussen (Rasmussen, 1986) will be used. Typically, the semantics increase when going higher into the model:

- On the autonomous layer, information is represented in the form of *signals* or simple values.
- On the associative layer, information is represented in the form of *signs*.
- On the cognitive layer, information is represented in the form of *symbols*.

In our cognitive architecture, the autonomous layer is not explicitly modelled. For that reason, we automatically translate what is perceived in the environment into signs. That means that only the translation from signs to symbols is needed.

5.2.1.3.1 Requirements

Several requirements on the translator can be defined (Rasmussen, 1986):

- the translator translates signs into symbols
- for all symbols, there is a way that they can be derived from signs or other symbols

To be able to implement the translator, the following requirements must be met:

- there must be a set of signs derived from perceiving the environment
- there must be a set of symbols
- there must be knowledge on how to translate both signs and symbols into symbols

5.2.1.3.2 Basic Idea

The translator is situated in the cognitive layer. Its task is to translate signs into symbols. There are thus two different sets of 'dictionaries', which are mutually exclusive. A sign cannot be a symbol or the other way around. The first 'dictionary', namely the signs, can be accessed by the associative layer and by the cognitive layer. The signs are the knowledge items that, in our architecture, are directly perceived from the environment, without introducing any interpretation. The second 'dictionary', the symbols, can only be accessed by the cognitive layer, which also produces the words in it through the translator. For the translation, extra knowledge is necessary.

To make the difference between signs and symbols explicit, the domain is modelled by two ontologies. We have two different ontologies, namely the ontology that specifies the signs that can be used by both the associative and cognitive layer and the ontology that specifies the symbols. The translator that translates signs into symbols is rule-based. These rules are reasoning rules, and no actions or goals derive from them. In this way, they differ from the GOMS rules used generate behaviour on the associative layer.

The role of the ontologies is to structure the domain and make explicit what the signs and what the symbols are. Thus, the ontologies are only a specification of a conceptualization, and do not concern reasoning.

The functionalities of the translator are as follows:

- *Signs to symbol translation* is needed to raise the level of abstraction of the signs present in the associative layer. By using symbols representing abstract concepts, we can simulate more abstract reasoning patterns in the decision-making mechanism. These translations can be regarded as *classification rules*, stating which observables (the signs), can be regarded as which types of situation (represented in symbols). This part can be seen as 'Identifying the situation' as described in Rasmussen's model (see Section 5.1).
- *Symbolic reasoning* is needed to represent reasoning patterns that have as input symbolically coded knowledge, and as output symbolically coded knowledge. These rules can be used to capture the background knowledge of the pilots, which they have acquired during their training and operational experience. In Rasmussen's model, this is the 'Judge and Evaluate' part of the cognitive layer (see Section 5.1).

5.2.1.3.2.1 Ontologies

The ontologies of signs and symbols are made explicit in the HUMAN project. The ontologies makes it possible to describe all procedures and interpretations of situations that are necessary to fly the scenarios.

These ontologies serve the following purposes:

1. for each symbol, there should exist at least one rule that translates signs/symbols into this symbol
2. it defines what are the signs (and thus is accessible from the AL) and what are the symbols (and thus only accessible from the CL)
3. to make syntactic and semantic assumptions of the domain explicit to facilitate implementation and communication
4. to check that a concept is not both a sign and a symbol, but belongs only to one of the ontologies.

The ontologies are created using an ontology editor Protégé (Fridman Noy, Fergerson, & Musen 2000). They are stored in the CLIPS Object Oriented Language (COOL) format. In this way, they can be conveniently loaded into CLIPS and be used as a basis for rule-based reasoning.

5.2.1.3.2.2 Sign-symbol translation and symbolic reasoning

For representing the reasoning required for sign-symbol translation and symbolic reasoning, we build on a rich tradition in AI, namely that of *rule-based* knowledge systems. This entails that we store knowledge in the form of rules which take the basic form of:

A -> B

The left part of the rule (A) is called the antecedent, and the right part of the rule (B) the consequent. Besides rules, a rule-based Knowledge-based system (KBS) contains *facts*. If a fact matches an antecedent of a rule (for example, if fact A is the case), we say that the rule *fires*. When a rule fires, its consequent is added to the fact base (in our example, fact B would be added to the fact base).

The rules in the translator have a different representation than the rules in the associative layer. As the rules' function is to translate knowledge from one representation of facts to another representation of facts (the translating from signs to symbols) and to reason about information, the consequent can only be other information. This differs from the rules in the associative layer, where the consequent can be goals, actions and information. For example, in the translator, it is not possible that a goal is made active. The evaluation of whether a goal should be executed or even whether it could be executed is done in the goal management, which uses the information produced by the translator, but is independent from the translator.

Both sign-symbol translation and symbolic reasoning are implemented in the form of rules. The difference is in the way these rules are formed. We can characterize the difference between sign-symbol translation rules and symbolic reasoning rules as follows.

- In sign-symbol translation rules, the antecedent is specified in terms of the ontology of signs; the consequent is specified in terms of the ontology of symbols.
- In symbolic reasoning rules, both the antecedent and the consequent are specified in terms of the ontology of symbols.

To implement the rule-based reasoning, we use CLIPS. CLIPS is integrated in Java using the Java Native Interface (JNI). In this way, we can make use of CLIPS reasoning power from within the cognitive architecture, implemented in Java.

We can divide the CLIPS rules in different libraries. In this way, we can control which *part* of the rule-base is processed for reasoning. This can be used to model efficient reasoning, i.e. reducing cognitive load by only deriving those pieces of information which are needed in the current situation. In CLIPS, we can implement this by only executing those sets of rules which lead to some piece of information.

5.2.1.4 Monitoring the Associative Layer

A task of the cognitive layer is to monitor the associative layer. This can be done in different ways.

First, the idea is to have some kind of mechanism that monitors the actions, goals or results of the associative layer to decrease the number of errors. On the cognitive layer, more knowledge is available (the symbols + extra knowledge on background and textbook knowledge) and better decisions can be taken. That means that if the AL uses a standard procedure to act in a situation, the cognitive

layer should be able to decide whether this is possible and appropriate, or whether the cognitive layer needs to take over.

This can be done in several ways:

1. monitoring of AL goals. This means that in particular situations, it is checked by the CL whether the AL has made a particular goal active. The CL knows that the AL can handle the situation by executing a standard procedure, but checks whether the AL has perceived the situation correctly and recognized that this procedure needs to be executed. This is done by checking whether the appropriate goal is active in the goal agenda of the AL.
2. monitoring dangerous situations that occur through errors on the AL. These kind of situations are recognized because of additional information from the translator.
3. checking the results of the associative layer.

When for instance an action is desired by the perception component (e.g., perceive a red light) the cognitive layer can check if the associative layer is working on the right goal. If it is not, the CL can

- set the goal in the goal agenda of the AL
- execute the task itself.

Another situation in which the cognitive layer can monitor the associative layer is when a goal/plan is provided by the cognitive layer to the associative layer and the cognitive layer keeps track of the final (or intermediary) goal (descent to 3000ft) and waits till the goal is perceived and when nothing changes alarms the associative layer.

5.2.1.5 Cognitive lockup

Cognitive lockup is the EPM that is implemented in the cognitive layer for the first cycle. In this section we explain how this EPM is implemented within the cognitive layer.

5.2.1.5.1 Basic Idea

Cognitive lockup is defined as a reluctance to switch tasks. This reluctance to switch can have different causes (Meij, 2004):

Planning

1. People commit themselves too early to a detailed plan (time is already invested);
2. People refrain from monitoring the environment;
3. People generate future scenarios that are too optimistic;

Task-switching

4. People lack spare attention to switch to a second disturbance;
5. The costs of switching are perceived too high;

Decision-making

6. People are loss averse: they weigh losses larger than gains;

7. Sunk costs: prior investments are taken into account;

8. Task completion: people have a desire to fulfil a task

We will concentrate on the task-switching aspects, which means that attention is not allocated to another task because the costs of switching are perceived too high. Of course the start priorities can cause a wrong prioritization too, but while we deal with experts we assume that the priorities for the normative activities are correct. The priorities of experts can differ also, but will be correct in essence. Cognitive lock-up happens when there are too many or not enough tasks. When there are many tasks, cognitive lock-up prevents people to change too often, and keep the attention with the task one is currently executing. This can lead to staying with tasks too long or with tasks that are less important than other tasks.

In the goal management, for all top-level goals we have defined a priority to be chosen for execution. This static priority is determined by a domain expert. In goal management however, the priorities are dynamically calculated taking also the situation into account, e.g. for how long a goal has been active but not selected to be executed. In the goal management's control mechanism, a decision mechanism is implemented that determines which of the active goals should be selected to be executed. This is done dependent on the priorities of the goals (see section 5.2.1.1). This mechanism is extended by the parameter *Task Switching Costs* (TSC), which determines the difference that the priorities need to have to halt the execution of a goal to select a different goal to be executed. Task Switch Costs are described extensively in literature (e.g. Liefoghe, Barrouillet, Vandierendonck & Camos (2008); Jersild (1927); Rogers and Monsell (1995); Monsell (2003)). The TSC depends on the number of tasks that is active and on the time to spare. The higher the number of active tasks is the higher are the costs to switch a task. Furthermore, the TSC is higher when there is a lot of time or little time to spare. Little time to spare happens when there are many tasks, but also when there are tasks that take a long time. A lot of time to spare also leads to increased TSC caused by boredom. The following formula determines the TSC:

$$\begin{aligned} \text{TSC} = & \text{StartTSC} \times (\text{Mean number of active tasks on cognitive layer in a minute}) \\ & + (\frac{1}{2} \times \text{Mean number of active tasks on associative layer in a minute}) \\ & + \text{factor boredom or stress dependent on idle time).} \end{aligned}$$

The parameter StartTSC denotes the threshold that gives the difference in priority two goals need to have to make an interruption of the one goal and the changing to the other goal possible. Only if the second goal has a priority that is at least the StartTSC higher than the priority of the first goal, a switch takes place. The parameter StartTSC should be determined by experimentation.

The higher the TSC is, the lower is the probability that the goal is switched. To determine whether a goal should be interrupted and a different goal should be executed, the current task priority is added to the TSC. Only if a priority of another goal is above this threshold, the other goal is chosen. This means that the more goals are active on the cognitive and on the associative layer or when there is no

time or a lot of time to idle, the higher the difference needs to be between the current goal and a new goal, even though the new goal might have a distinctly higher priority than the current goal.

5.2.2 Inter-layer communication

The associative and the cognitive layer interact. Several different kinds of communication can be identified:

- The monitoring of the associative layer by the cognitive layer. As described in section 5.2.1.4, the cognitive layer can check whether a goal is active on the associative layer, and if not, add this goal to the goal agenda in the associative layer. In addition, monitoring includes that the cognitive layer checks (some) results of the associative layer. This is implemented through implementing a request by the associative layer to the cognitive layer to evaluate a result, before further proceeding with this result.
- The associative layer can realize that it does not have a procedure to handle the situation. In this case, the associative layer calls on the cognitive layer to handle the situation. As the cognitive layer has more background and textbook knowledge, the goal is handled and executed on the cognitive layer.
- The cognitive layer can activate goals on the associative layer. This is for example the case if a goal on the cognitive layer is implemented by several other sub-goals, and one of these sub-goals is a procedure. In this case, the cognitive layer activates a goal on the associative layer, which leads to the execution of a procedure by the associative layer.

5.2.2.1 Example of the inter-layer communication

In this section, we describe an example scenario in which the communication and different tasks of the layers is depicted in more detail.

5.2.2.1.1 Example Scenario Storm Avoidance

To demonstrate the functionality of the cognitive layer and the associative layer and the interaction between them an example is described. During the flight a storm on the trajectory appears on the AHMI. The pilot has to replan in order to fly around the storm.

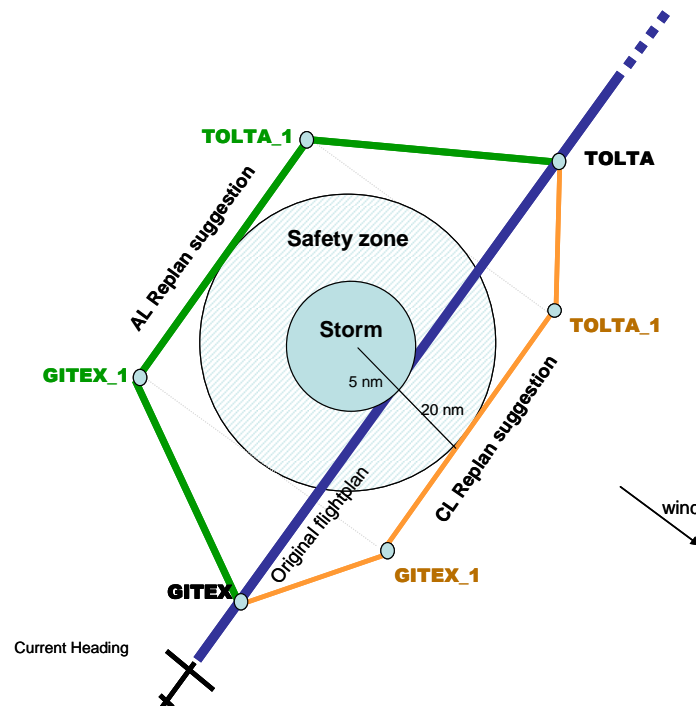




Figure 5-21 Example scenario. Blue line: original trajectory. Green line: trajectory suggested by AL after replanning. Orange line: trajectory suggested by CL after replanning.

To fly this scenario, the following goals are implemented in the cognitive layer:

- EvaluateRerouting (retrieves the suggested rerouting from the working memory and places it in the SST)
- CognitiveReasoning (reasons about the flightplans in the SST)
- CognitiveReplan (cognitive steps generate a new flightplan)
- ExecuteIntendedPlan (retrieves the favourable flightplan from the SST and places it back into the working memory)
- HandleAtcCall (handles an incoming ATC call)
- AssessMemoryContent (looks for changes in relevant variables in the working memory)
- Idle, sleep goal (if no goal is active, this goal is chosen)

The following steps take place in this scenario:

Step	Layer	Description	AL plan	CL plan	Goal on AL	Goal on CL
1	AL	The AL produces a normative flightplan for taking a detour: the Green Upwind (Green) detour. This is done according to standard procedures.	Green Upwind Detour (intended, not deliberated)		AL Replan	

	HUMAN Model-based Analysis of Human Errors during Aircraft Cockpit System Design	
---	--	---

2	CL	Retrieve the rerouting and storm from the working memory. Translate signs into symbols.				Evaluate ReRouting
3	CL	The Decision Making Module chooses to perform sign symbol translation, because there is a storm involved (i.e. the followed route deviates from the currently planned route)				Cognitive Reasoning
4	CL	The SST labels the flightplan: the upwind route is unfavourable because it is much longer than the currently planned route.	Green Upwind Detour (intended, deliberated , unfavourable)			
5	CL	The Decision making module chooses to activate the cognitive planner .				Cognitive Replan
6	CL	The cognitive planner creates a short plan (the red plan).	Green Upwind Detour (not intended , deliberated, unfavourable)	Red Downwind Detour (intended, not deliberated)		
7	CL	The Decision Making Module chooses to perform sign symbol translation.				Cognitive Reasoning
8	CL	The SST evaluates the new CL plan to be good.		Red Downwind Detour (intended , deliberated , favourable)		
9	CL	The Decision Making Module chooses to execute the intended favourable plan (= Red)				Execute Intended Plan
10	AL	The AL executes the Red plan (the downwind plan).			AL Implement Plan	

Table 5-1 Steps that describe the execution of the cognitive model for the scenario

This scenario shows the interaction between the associative and the cognitive layer. A storm is perceived on the screen, and the associative layer plans a new trajectory around the storm. The cognitive layer monitors the associative layer to see whether it should take control and evaluate actions or results on a more conscious level. This is done when realizing that the associative layer has thought of a new trajectory. The cognitive layer assesses the replan-suggestion of the associative layer. In the scenario described above, the cognitive layer evaluates the new trajectory to be not favourable, and starts a replan-goal on the cognitive layer. On the cognitive layer,

there is more background knowledge available and non-proceduralized decision-making. A new, different, plan is generated to avoid the storm. This plan is evaluated by the cognitive layer and marked as favourable. The plan then is passed on to the associative layer to implement in the AHMI. The cognitive layer thus delegates proceduralized tasks to the associative layer.



5.3 Status of the Cognitive Architecture

The following table summarizes the acceptance conditions for the cognitive model, as defined in D3.1 – “Acceptance Conditions for the virtual simulation platform”, and their assigned status, as decided by the industrial partners AIF and ALA during the acceptance tests on June 16th. 2009. The compliance to the acceptance conditions has been evaluated by means of checks and tests (definition from D3.1):



- *Checks* typically mean directly verifying specific properties, without executing simulated runs on the Virtual Simulation Platform. Checks are typically performed by domain experts (i.e., simulation experts, system experts, pilots). Checks are optional (i.e., there may be no check for a given acceptance condition).
- *Tests* typically mean performing active tests on the Virtual Simulation Platform, by means of short simulated runs, aimed at testing specific properties. Tests are optional (i.e., there may be no test for a given acceptance condition).

Acceptance Condition	Sub Req	Description of each requirement	Check/Test status
VSP AC.1-1 Complete and correct implementation of the cognitive model	1	The VSP must be able to simulate a crew, consisting of PF and PM (or PNF). Therefore the platform must support communication between the PF and PM, restricted to simulated verbal communication.	Check: Accepted
			Test: Accepted
	2	The VSP must be able to interpret the crew (PF, PM) normative procedures.	Check: Accepted
			Test: Accepted
	3	The VSP must be able to indicate the actor performing the actions/procedure tasks (PF, PM) in the results presentation	Check: Accepted
			Test: Accepted
	4	The perception of the CM must be sophisticated, in terms of a restricted visual field with focus, as well as eye-movements. Perception must support the detection of	Check: Accepted
			Test: Accepted

		certain modelled events, e.g. warning lights or sounds, which trigger reactive rules.	
	5	The CM must support "Loss of information in working memory" (OPTIONAL)	Check: Partially accepted
			Test: Partially accepted
	6	The Associative Layer must support multitasking, in terms of maintaining multiple goals at the same time and switching between them, as well as interrupting a goal to deal with reactive goals.	Check: Accepted
			Test: Accepted
	7	The Associative Layer must support reactive behaviour, in terms of reacting on external events of visual or vocal type, limited to a set of events and rules, modelled in the procedure and/or a dedicated file	Check: Accepted
			Test: Accepted
	8	The Cognitive Layer must have a goal hierarchy, connected to the goal hierarchy of the Associative Layer	Check: Accepted
			Test: Accepted
	9	The Cognitive Layer must be able to translate signs into symbols.	Check: Accepted
			Test: Accepted
	10	The Cognitive Layer must be able to monitor the Associative Layer or the actions of the associative layer in the environment.	Check: Partially accepted
			Test: Partially accepted
	11	The Cognitive Layer must implement "Attentional Capture".	Check: Accepted
			Test: Accepted
	12	The Cognitive Layer must implement "Salience Bias" (OPTIONAL)	Not Implemented
	20	"Learned Carelessness" must be implemented.	Check: Partially accepted
			Test: Partially accepted

	HUMAN Model-based Analysis of Human Errors during Aircraft Cockpit System Design	
---	--	---

VSP AC.2 Complete and correct implementation of the A/C model on the Virtual Simulation Platform	1	The A/C model must provide all information needed by the CM in order to fly the scenarios.	Check: Accepted
	2	The A/C model has to be available for the VSP during WP3 for testing purposes, as well as for WP4.	Test: Accepted
	3	Dynamic tests of the ability of the simulated crew to interact with the A/C.	Check: Accepted
VSP AC.3 Complete and correct implementation of the environment model on the Virtual Simulation Platform	1	The environment model (weather- and traffic-simulation) must provide all information needed by the CM in order to fly the scenarios.	Test: Accepted
	2	The environment has to be available for the VSP during WP3 for testing purposes, as well as for WP4.	Check: Accepted
	3	Dynamic tests of the ability of the simulated crew to interact with the environment (weather and other traffic)	Test: Accepted
VSP AC.4 Complete and correct implementation of the target system (AHMI) model on the Virtual Simulation Platform	1	The target system must provide all information needed by the CM in order to fly the scenarios.	Check: Accepted
	2	The target system has to be available for the VSP during WP3 for testing purposes, as well as for WP4.	Test: Accepted
	3	Dynamic tests of the ability of the simulated crew to interact with the target system (AHMI).	Check: Accepted
VSP AC.5 Complete and correct implementation of the Virtual Simulation Platform infrastructure	1	Software Tools have been provided by DLR (A/C model, env., datapool, ...).	Test: Accepted
	2	Remote Software available for testing and for WP4	Check: Accepted
			Test:

	<p style="text-align: center;">HUMAN</p> <p style="text-align: center;">Model-based Analysis of Human Errors during Aircraft Cockpit System Design</p>	
---	---	---

			Accepted
	3	The VSP must be able to communicate with the GECO software via the datapool.	Check: Accepted
			Test: Accepted
	4	The VSP must be able to run in both manual and batch modes (optionally it would be nice to have the possibility of accelerating the speed of the runs).	Check: Partially accepted
VSP AC.6 Reliability and availability of the Virtual Simulation Platform			Test: Partially accepted
	1	Reliability and bug management policy.	Check: Accepted
	2	Short tests	Test: Partially accepted
	3	Known bugs	Check: Accepted
VSP AC.7 Understandability and documentation of the cognitive model	4	Long test sessions	Test: Not accepted
	1	Documentation of CASCaS	Check: Partially accepted
VSP AC.8 Scalability and support to incremental developments of the cognitive model	1	VSP architecture adequate?	Check: Accepted
	2	CA architecture adequate?	Check: Accepted
VSP AC.9 Compliance with future experiments and experimental scenarios	1	Compliance with future experimental scenarios.	Check: Partially accepted
			Test: Partially accepted
VSP AC.10 Data logging capabilities and data formats	1	Appropriateness of the data logged.	Check: Accepted
	2	Overall integration of the VSP with the data analyses tools.	Test: Partially accepted
	3	Appropriateness of data formats	Check: Accepted
VSP AC.11	1	Principles and solutions for ensuring functional	Check:
06/07/2009			Named Distribution Only
			Page 71 of 86
			Proj. No: 211988

Functional equivalence with the Physical Simulation Platform		equivalence.	Accepted
	2	Functional equivalence tests.	Test: Partially accepted
VSP AC.12 Support to acceptance evaluation process	1	Measures put in place by OFF and TNO to support the acceptance evaluation process.	Check: Accepted

Table 5-2 Results of the Acceptance Tests for the VSP

All partially accepted and not accepted test conditions will be fixed until 31st July 2009. The full result and comments to the partially accepted conditions can be found in Appendix II – “Acceptance Tests VSP Results”.

6 Virtual Simulation Platform

The following section describes two tools, which are needed for the VSP. The first one is the scenario controller, which is also used for the PSP, and the second one the Symbolic AHMI.

6.1 Scenario Controller

In this section, the scenario controller is described in more detail. The objective of the scenario controller is to start and control the simulation in accordance to the scenario definition. This means to trigger events (e.g. system failures) and ATC communication, and to stop the simulation if the scenario end has been reached. The scenario controller has to be able to do this for the VSP and for the PSP.

In the following, requirements for the scenario controller should be derived, in order to define the features and capabilities of the controller.

6.1.1 Requirements for the Scenario Controller

There are several requirements that can be identified for the scenario controller:

1. The scenario controller has to start the simulation. This includes
 - a. Adding the initial data to the data pool at t_0 . This initial data includes
 - i. Starting time in HH:MM format
 - ii. Own aircraft properties (also with a range to choose from, e.g. initial altitude between FL190 to FL220 on some of the variables. Maybe also together with a probability distribution for the range)
 1. Speed;
 2. Heading;
 3. Altitude;
 4. Position;

5. Active Constraint list and active trajectory (as we start in flight, this should be set up by the controller), e.g. the AHMI should be on an active trajectory.
 - iii. Environment properties
 1. Wind direction;
 2. Wind speed;
 3. Temperature;
 4. Visibility conditions
 - a. Clouding : e.g. overcast (8/8), broken (5-7/8), scattered (3-4/8), few (1-2/8), sky clear (0/8)
 - b. Sight : e.g. Km
 - b. It has to be made sure that the scenario is synchronised with the cognitive model.
 2. The scenario controller needs to end the simulation. This includes
 - a. Deciding when the scenario is ended.
 - b. Determining a success/failure condition and communicating this to the cognitive model. This is needed for the AL to be able to adjust their rules to elicit learned carelessness.
 - c. As learned carelessness has to be facilitated, after realizing that the scenario is ended, the new simulation run may only be started after having made sure that the cognitive model (more specifically, the associative layer) has had the time to adapt its rules.
 3. The scenario controller must be able to trigger events. These events must not only be dependent on time, but can relate to e.g. actions of the pilot, situations in the environment or other events. The following events must be possible (not exhaustive):
 - a. System malfunctions
 4. The scenario controller has to be able to simulate ATC communication. This includes both reacting to an ATC call from one of the pilots, and taking the initiative and start communication with the pilots. For reacting to communication from the (co-)pilot, the following aspects have to be taken into account:
 - a. The scenario controller needs to be able to 'understand' and react to all relevant possible communication by the pilots. Which communication is relevant within the scenarios will be determined during scenario specification. Relevant communication is e.g. about clearances (descend, climb, heading, speed), weather information, traffic information, traffic sequencing, and route modifications. In Controller-Pilot Data-Link Communications (CPDLC), this is done in a similar way just that one gets a handover text message which is to be acknowledged by a soft-button. This greatly relieves the frequencies and the CPDLC unit can handle up to 200 commands at a time. The ATC communication needs to be formalized, as it has to be made sure that simulations run in the VSP and the PSP are comparable.

5. The scenario controller must be able to simulate other traffic (in terms of starting a predefined traffic simulation).
6. The scenario controller has to be able to simulate weather. Weather can be distinguished in pre-recorded weather (wind speed and wind direction) and thunderstorms.
7. The scenario controller should be implemented in a way that assists scenario definition.
8. The scenario controller must be able to realize the above mentioned requirements for both the VSP and the PSP and ensure comparability.

6.1.2 Model of the Scenario Controller

In this section, it is described how the scenario controller is realized to fulfil the requirements identified in section 6.2.1.

Several modules can be identified that make up the scenario controller, as depicted in Figure (grey components are existing components from DLR): In the following sections, these components are described in detail.

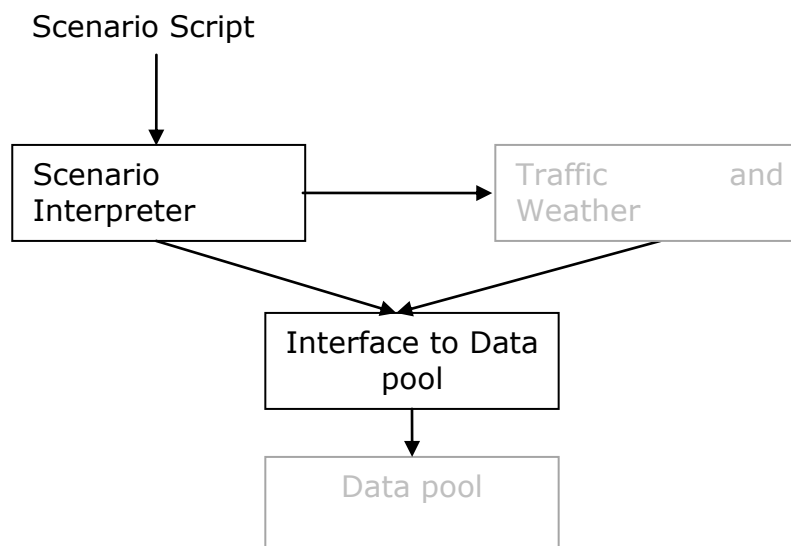


Figure 6-1: Scenario Controller Components

6.1.3 Scenario Script

The Scenario Script is the formalization of the scenarios. It has to make sure that first of all, it is easily understandable for subject matter experts so that they can define scenarios.

Second, the scenario script has to include all necessary aspects of the scenario, which includes that the following has to be specified.

The scenario is specified in a CLIPS file, which is a rule-based system that allows storing facts, such as the current trajectory, but also rules that trigger events, e.g. an ATC call when the aircraft is at a specified waypoint. For an introduction to CLIPS, see <http://clipsrules.sourceforge.net/WhatIsCLIPS.html>. For further documentation see (Lyndon B. Johnson Space Center (1993)) and (Lyndon B. Johnson Space Center (1993b)). CLIPS has also been used for implementing the sign-symbols translator. See section **Fehler! Verweisquelle konnte nicht gefunden werden.** above.

6.1.4 Traffic and Weather Module

The DLR has a tool that can re-play traffic and weather simulations that have been recorded beforehand. This means that on the PSP, traffic and weather are simulated, and the transactions with the data pool (or better: the according data in the data pool) are recorded. This recording can later be played, which means that the corresponding data is set into the data pool at the right time.

As a consequence, the traffic and weather simulation is static and thus independent of the scenario script. Therefore several traffic and weather simulations have to be specified and recorded beforehand. In the scenario script, it is only specified which traffic and weather simulation run is going to be part of the current simulation run. The scenario interpreter reads and interprets the scenario script, and invokes the according weather and traffic recording in the traffic and weather module, which starts the replay.

The following aspects are included into the specification of weather and traffic:

- the density of the traffic
- the flight path of the planes
- wind speed and direction
- thunderstorms
 - o Position
 - o Size
 - o Wind speed and direction

6.1.5 Scenario Interpreter

The scenario interpreter reads the scenario script and interprets the content of the script. Its task is to control the simulation run. An important task is to start the scenario execution. For this, the following actions have to be taken:

- The initial data has to be posted to the data pool. This is done via the interface to the data pool
- The traffic and weather module has to be called with the specification of which traffic and weather recording has to be played. It has to make sure that the traffic and weather module starts its playing of the recording at the same time as the scenario starts.

- It has to be communicated to the cognitive model that the scenario starts.

During the execution of the scenario, for each possible event, the scenario interpreter needs to read the data from the data pool regularly (refresh frequency is 2m ms) to evaluate whether a condition of an event evaluates to true. If this is the case, this event has to be triggered and the corresponding data has to be set into the data pool. Reading and writing the data is done via the interface to the data pool.

The ATC functionality is included in the scenario interpreter. This task includes both reacting to an ATC calls from the (co-)pilot and initiating communication itself. In the first case, it has to be made sure that for all relevant possible communication, it can react appropriately. The interpreter decides, according to defined rules in the scenario script, whether an ATC event needs to be triggered. If the pilot decides to communicate with the ATC, this is read from the data pool and an answer by the ATC is generated. For the moment, it has been decided that every trajectory that the pilot negotiates with the ATC will be accepted, as generating a new counterproposal goes beyond the scope of the scenario interpreter.

In the scenario script, a condition when to end the scenario is specified. This is read by the scenario interpreter. The scenario interpreter evaluates whether the simulation run has ended. If this is the case, and it was a run on the VSP, the scenario interpreter determines whether the simulation has been executed correctly, and posts a success or failure event to the data pool (via the data pool interface), which in turn can be read by the cognitive model to adjust its rules. It then waits for data/a message from the cognitive model that this has been concluded, and that a new run can start. Then, the data in the data pool is deleted and a new script is read.

6.1.6 Interface to the Data pool

All modules interact with the data pool via this interface. The following methods must be available:

- read data
 - o timestamp
 - o all environment variables (e.g. wind direction)
 - o all aircraft variables (e.g. heading and altitude)
 - o all interface variables
 - o communication messages from A/C
 - o message from the cognitive model that it has finished the simulation, also message that says that the model has finished updating its rules after having received the success/failure event
- write data
 - o success/failure of scenario
 - o all events (ATC messages and System messages)

- communication messages by the ATC module (negotiation downlink)
 - traffic/weather data
- start simulation. This has to make sure that all modules are synchronized.
- end simulation. The cognitive model is stopped.

6.1.7 Status of the Scenario Controller

The scenario controller has been designed and implemented in co-operation with TNO and OFF. The scenario controller has not been part of the acceptance conditions for the VSP. Currently most functions for the first cycle have been implemented, such that the use in the experiments is possible. Currently, the scenario scripts for the scenarios as defined in WP2 are under specification.

6.2 Symbolic AHMI

In this section, the Symbolic AHMI (SAHMI) is described in more detail. There are two objectives of this tool:

- In the current implementation of the AHMI, it is technically not possible to gather all information that is needed by the cognitive architecture (CA) via the datapool interface. Thus, one objective of the SAHMI is to provide the missing information for the cognitive architecture.
- A second objective is to perform an online and offline usability evaluation of the AHMI. This analysis provides e.g. the cognitive load the AHMI design causes. An overview of the SAHMI was introduced in D1.4 – “Redaction of New Standards relating to AHMI Designs”.

The tool chain, including the CA, is depicted in Figure 6-2. The task of the SAHMI is to monitor and send messages, related to AHMI interaction, to the CA. This means that in order to execute an action on the AHMI, the CA sends a message to the data pool indicating the triggered event (e.g. click on negotiation button). Then, the SAHMI reads the message from the Datapool and analyze its content in order to provide a feedback to the CA. There are two tools that are sources for the current state of the AHMI, the datapool, and the CHIME tool. CHIME is mainly used to connect and synchronize multiple instances of the AHMI (e.g. PF AHMI and PM AHMI) and the AFMS. Based on the action to perform, the status of the CHIME, and the status of the Datapool, the set of action is triggered internally (e.g. change state of the system to indicate negotiation) to update the CHIME and Datapool. Finally the feedback, a set of messages with information related to the visual feedback resulting from this action, is sent to the Datapool. The messages are accessed by the CA.

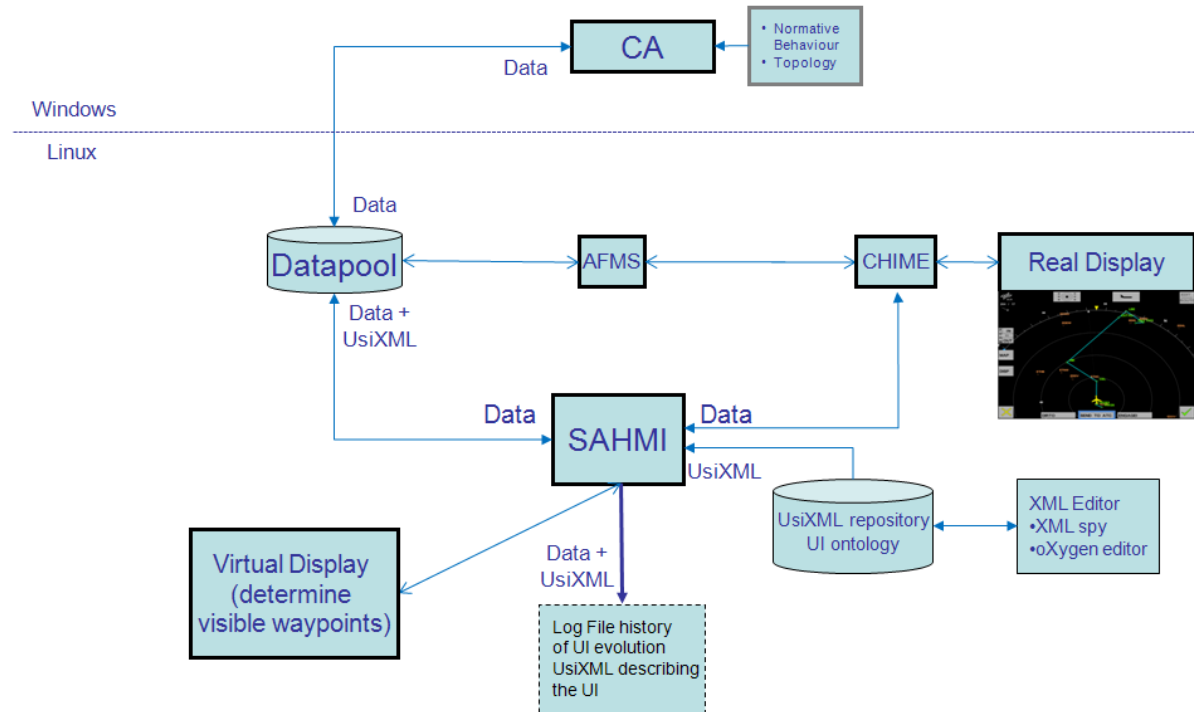


Figure 6-2: Symbolic AHMI Tool Chain

In the following, requirements for the SAHMI should be derived, in order to define the features and capabilities of the system.

6.2.1 Requirements

The following list the requirements identified for the SAHMI:

1. The SAHMI has to be coherent and compatible with the physical AHMI, this includes
 - a. Reflect system states from the physical AHMI (CHIME and Datapool) to capture and store them in data structures, this includes.
 - i. Layout distribution
 - ii. Enabled controls (buttons, menus, toggle buttons)
 - iii. Visible controls (buttons, menus, toggle buttons)
 - iv. Trajectory status (selected, negotiated, actual)
 - v. Waypoints
 - vi. Constraint list
 - vii. Visible widgets
 - viii. Visible regions
 - ix. Visible objects (airports, aircrafts, enroute-airways, nav aids, SID, star approaches)

- x. Visible option on trajectory (constraint list, restricted areas, cas, active trajectory, weather radar)
 - xi. View status (vertical/horizontal)
 - xii. Mode status (plan/monitor)
 - xiii. Zoom factor (increment, max, min, current value)
 - xiv. Suitable waypoint based on the current constraint list
 - xv. Selected waypoint
 - xvi. Current waypoint
 - xvii. Next waypoint
 - xviii. Previous waypoint
 - xix. Next waypoint
 - xx. Last overflown waypoint
 - b. Reflect system behaviour, particularly the one producing visual feedback (e.g. new trajectory).
 - i. Colour change
 - ii. Messages
 - iii. Show/ hide widget
 - iv. Show/ hide object (airports, aircrafts, enroute-airways, nav aids, SID, star approaches, constraint list, restricted areas, cas, active trajectory, weather radar)
 - v. View rotation
 - vi. View translation
 - vii. Objects rotation
 - viii. Objects translation
 - ix. Trajectory calculation
 - x. Visible objects
 - c. Events handling
 - i. Mouse click
 - ii. Mouse over
2. The SAHMI must use a client interface to the Datapool, monitoring message communication with the CA, this includes
- a. Adding messages to the Datapool. A message exchange queue data structure is needed to write messages.
 - b. Message format is needed. This message format must be compatible with the message format use by the CA to assure interchange of messages between the SAHMI and the CA.
 - c. Reading messages. A thread constantly retrieving messages from the CA that are stored in its corresponding queue structure in the Datapool.

The SAHMI must be able to realize the above mentioned requirements for the VSP and ensure comparability.

6.2.2 Implementation status

In Figure 6-3, the SAHMI architecture is shown. A repository with a UsiXML formalism describing the AHMI is used. These specifications can be edited using any text editor; a XML-editor is preferable but not mandatory. This file is read using a parser that validates the specification and transforms this into a machine readable structure called model merger. The user interface is complemented with data accessed from the data pool or the CHIME. From the Datapool, messages from CA are retrieved while the CHIME client collects information about the status of the system. Both sources of information are processed in the model merger and a message for the CA is sent to the Datapool client. Finally the data from the data pool and the CHIME must be transformed to be compatible with UsiXML format in order to store a log file history of UI evolution to be used for UI evaluation.

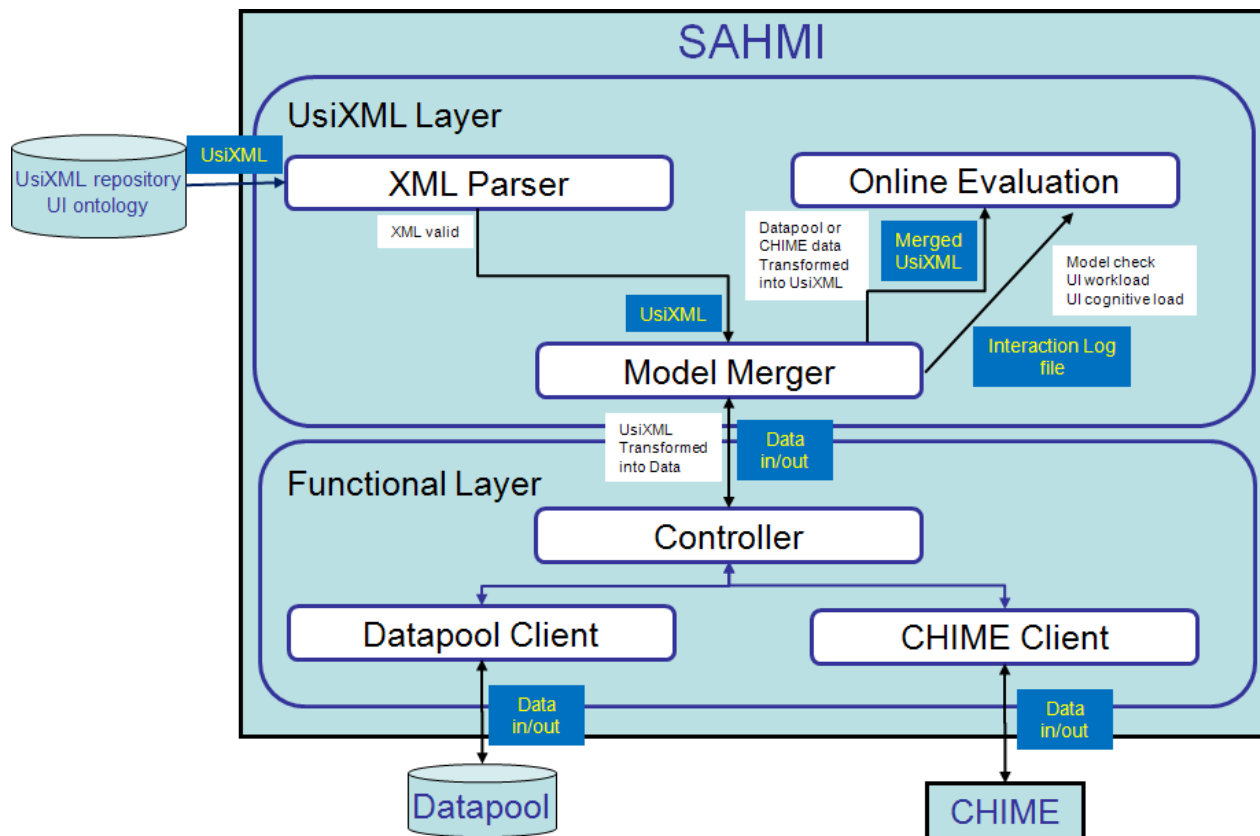


Figure 6-3: Symbolic AHMI Architecture

7 Summary

Objective of this document was to describe the Virtual Simulation Platform (VSP), with focus on how the requirements of D1.3 have been handled. In addition to this, the results of the acceptance tests on the test conditions that have been defined in D3.1 are described.

The VSP consists of several tools. Main focus in this document was the Cognitive Architecture, and on the tools that have been exclusively implemented for HUMAN, namely the Scenario Controller and the Symbolic AHMI.

For the cognitive architecture CASCaS, several requirements have been defined, with the objective to enable the cognitive crew to fly the scenarios that have been defined in WP2. Much effort has been invested to design and implement these requirements, and the acceptance conditions showed that this was achieved successfully. CASCaS has been extended with the following features:

- CASCaS has been extended for modelling a wider set of cognitive behaviour. This has been achieved in adding a cognitive layer and a autonomous layer to CASCaS, based on the theory of (Rasmussen 1987).
- The cognitive layer, designed and implemented by TNO, interacts with the associative layer, provided by OFF, and supports a large set of behaviour, beginning with a transformation from signs to symbols (deriving new information from known facts), to monitoring the associative layer, decision making, and reasoning. In addition to this, the EPM "Attentional Capture" has been implemented on the cognitive layer, too.
- A speech motor and an auditory component, to allow crew coordination between multiple instances of CASCaS, has been modelled and implemented.
- The percept and motor component have been extended and improved, in order to show a more sophisticated and natural behaviour. This includes the implementation of a focus, a visual field and eye movements, as well as Selective Attention. The motor component now enables a richer set of actions, with a grounded theory of timing for the actions. Also hand-eye coordination has been implemented.
- CASCaS now allows multi-tasking on the associative layer, thus allowing the associative layer of working on multiple goals at the same time, and switching between them.
- The working memory has been extended to allow the EPMs "Loss of information in Working Memory" and "Learned Carelessness".

In order to acquire the normative behaviour that is needed by CASCaS to fly the scenarios, a chain of two tools has been set up. As input for CASCaS, a formal description of the tasks on a key-stroke level is needed, in order to fly the scenarios

with the cognitive crew. Current task-descriptions (if existing at all) are described mainly un-formal, e.g. as a text. In order to support the creation from an un-formal description to a formal procedure, we developed for HUMAN a set of two tools, which support this step. In the first tool, AMBOSS, a semi-formal description of the tasks of the crew is created from the un-formal description. This semi-formal description is then input to the second tool, PED. PED allows the formal description of tasks on a key-stroke level, and the output of PED can be directly used in CASCaS. The introduction of the semi-formal description as an intermediate step makes it easier for the modellers to come from the un-formal text to a very formal procedure.

As some information that are needed by CASCaS are not available in the tools that have been provided by DLR, it was necessary to develop a symbolic AHMI (SAHMI), that makes this information available and accessible for the cognitive crew. In addition to this, the SAHMI also analyses the user-interfaces, e.g. the cognitive load. The SAHMI uses a formal description of the user interface (AHMI) in the UsiXML language for the analysis. UsiXML has been proposed to become a W3C standard in the framework of HUMAN.

For the purpose of controlling the scenarios that have been defined in WP2, a scenario controller has been developed. Several requirements have been identified, and then transferred into an implementation. The scenario controller reads a scenario description in a rule-based language (CLIPS) and then starts events according to the scenario description, e.g. when the aircraft is at a certain waypoint.

8 References

- AMBOSS, 2009, URL: <http://mci.cs.uni-paderborn.de/pg/amboss/>
- Anderson, J. R. (1983). A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, 22(3), 261–295.
- Anderson, J. R. (2000), *Learning and Memory – An Integrated Approach*, 2nd Edition, John Wiley & Sons Inc., ISBN 0-471-24925-4
- Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.
- Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, 111(4), 1036–1060.
- Baron, M., Lucquiaud, V., Autard, D. & Scapin, D.L (2006): K-MADe: un environnement pour le noyau du modèle de description de l'activité. *Proceedings of the 18th Frenchspeaking conference on Human-computer interaction*, Montreal, Canada.
- Barthet, M.-F., & Tarby, J.-C. (1996). The Diane+ method. In J. Vanderdonckt, (Ed.), *Computer-aided design of user interfaces* (pp. 95–120). Namur, Belgium: Presses Universitaires de Namur.
- Basnyat, S. & Bastide B. (2006): *Error Patterns: Systematic Investigation of Deviations in Task Models*, TAMODIA, 2006, p. 109-121
- Biere, M., Bomsdorf, B. & Szwillus, G. (1999): Specification and simulation of task models with VTMB. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, May 15 - 20, 1999). CHI '99. ACM Press, New York, NY.
- Bolognesi, T. and Brinksma, E. (1987): Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1): 25-59, 1987.
- Card, S.K., Moran, T.P. and Newell, A. (1983): *The Psychology of Human Computer Interaction*, Lawrence Erlbaum Associates
- Connor, C. E., Egeth, H. E., & Yantis, S. (2004). Visual attention: bottom-up versus top-down. *Current Biology*, 14, 850— 852.
- Fischer, B. (1999). *Blick-punkte - neurobiologische prinzipien des sehens und der blicksteuerung*. Bern: Verlag Hans Huber.
- Folk, C. L., & Remington, R. W. (1994). The structure of attentional control: contingent attentional capture by apparent motion, abrupt onset and color.
- Freed, M. A. (1998). *Simulating human performance in complex, dynamic environments*. Unpublished PhD Thesis , Northwestern University.
- Fridman Noy, N., Fergerson, R., and Musen, M. (2000). The knowledge model of protege-2000: Combining interoperability and flexibility. In *Proceedings of EKAW 2000*.
- Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G. and von Detten M.: *AMBOSS: A Task Modeling Approach for Safety-Critical Systems*, LNCS, Springer-Verlag,

- Berlin, 2008. 7th Int. Workshop on Task Models and Diagrams TAMODIA'2008
<http://mci.cs.uni-paderborn.de/pg/amboss/>, ISBN: 978-3-540-85991-8
- GUITARE: EU-Project URL: <http://giove.cnuce.cnr.it/Guitare/index.html>
 - HUMAN Project Consortium, D1.4 Redaction of New Standards Relating to AHMI Designs, 2008.
 - Jersild, A.T. (1927) Mental set and shift. *Archives Psychology*, 89
 - Johnson, C.: Final report: Review of the BFU Überlingen Accident Report. http://www.dcs.gla.ac.uk/~johnson/Eurocontrol/Ueberlingen/Ueberlingen_Final_Report.PDF, 2004.
 - Kastner, S., and Ungerleider, L. G. (2000). Mechanisms of visual attention in the human cortex. *Annual Review of Neuroscience*, 23(1), 315— 341.
 - Kieras, D. E., and Meyer, D. E. (1997). An overview of the epic architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, 12(4), 391–438.
 - Laird, J. E. (2008): Extending the Soar Cognitive Architecture, In: AGI, Memphis, pp. 224-235.
 - Laird, J. E., Newell, A., and Rosenbloom P. S. (1987): SOAR: an architecture for general intelligence. *Artif. Intell.* 33, 1 (Sep. 1987), 1-64. DOI= [http://dx.doi.org/10.1016/0004-3702\(87\)90050-6](http://dx.doi.org/10.1016/0004-3702(87)90050-6)
 - Laird, John E. & Wang, Yongjia (2006): Integrating Semantic Memory into a Cognitive Architecture, http://msoar.org/publications/2006/integrating_semantic_memory_cognitive_architecture
 - Liefoghe, B., Barrouillet, P., Vandierendonck, A., Camos, V. (2008); Working memory costs of task switching, *Journal of experimental psychology: Learning, memory and cognition*, 34(3), 478-494.
 - Lu, S., Paris, C., Vander Linden, K: Tamot: Towards a Flexible Task Modeling Tool. In The Proceedings of Human Factors 2002, Melbourne, Australia, November 25-27.
 - Lyndon B. Johnson Space Center (1993): CLIPS Basic Programming Guide Version 6.0, <http://www.gsi.dit.upm.es/docs/clipsdocs/clipshtml/vol1.html>
 - Lyndon B. Johnson Space Center (1993b): CLIPS Advanced Programming Guide Version 6.0 <http://www.gsi.dit.upm.es/docs/clipsdocs/clipshtml/vol2.html>
 - Meij, G. (2004) Sticking to plans: Capacity Limitation Or Decision-making Bias? PhD Thesis Experimental Psychological Research School, University of Amsterdam.
 - Monsell, S. (2003); Task switching, *Trends in Cognitive Sciences*, 7(3), 134-140.
 - Mori, G., Paternò, F., Santoro, C.: CTTE: support for developing and analyzing task models for interactive system design. *IEEE Trans. Softw. Eng.* 28, 8 (Aug. 2002), 797-813.
 - Mumaw, R. J., Sarter, N. B., & Wickens, C. D. (2001). *Analysis of pilot's monitoring and performance on an automated flight deck*. Paper presented at 11th International Symposium on Aviation Psychology, Columbus, OH.

- Nikolic, M. I., Orr, J. M., & Sarter, N. B. (2004). Why pilots miss the green box: how display context undermines attention capture. *The International Journal Of Aviation Psychology, Lawrence Erlbaum Associates, Inc.*, 14(1), 39— 52.
- Nuxoll, A., and Laird, J. E. (2004): Comprehensive Working Memory Activation in Soar, In ICCM, 2004, pp. 226-230.
- Orasanu, J. (1993): Decision-making in the Cockpit, In: E. L. Wiener, B. G. Kanki, and R. L. Helmreich et al.: Cockpit Resource Management, Academic Press, S.137-172, 1993.
- Rasmussen, J. (1986). Information processing and human-machine interaction: an approach to cognitive engineering. New York, NY, USA: Elsevier Science Inc.
- Rasmussen, J. (1987). Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *System design for human interaction*, 291— 300.
- Rogers, R.D., Monsell, S. (1995); Costs of a predictable switch between simple cognitive tasks, *Journal of experimental Psychology – General*, 124(2), 207-230.
- Salvucci, D.D. (2005). A multitasking general executive for compound continuous tasks. *Cognitive Science* 29, S. 457-492.
- Sexton, J.B., & Helmreich, R.L. (2000). Analyzing cockpit communications: The links between language, performance, error, and workload. *Human Performance in Extreme Environments*, 5(1), 63-68.
- Shankar R. (1991): Z-Scheduler: Integrating Theories of Scheduling Behavior into a Computational Model. In: Proceedings of the IEEE Conference on Systems, Man, and Cybernetics, pp. 1219-1223
- Shannon, C. (1948): A mathematical theory of communication. *The Bell System Technical Journal*, 27, 379-423, 623-656.
- Shannon, C., & Weaver, W. (1949):. The mathematical theory of communication. Urbana, USA, University of Illinois Press.
- Shapiro, Stuart C. (1971): A net structure for semantic information storage, deduction and retrieval, *Proc. IJCAI-71*, pp. 512-523
- Shepherd, A.: Hierarchical Task Analysis, 1995, CRC,
- Sowa, John F. (2006): Semantic Networks, <http://www.jfsowa.com/pubs/semnet.htm>
- Stuart, J. & Penn, R. (2004): TaskArchitect: Taking the work out of task analysis. In Proceedings of the 3rd Annual Conference on Task Models and Diagrams (Prague, Czech Republic, November 15 - 16, 2004). TAMODIA '04, vol. 86. ACM Press, New York, NY, 145-154
- Szekely, P., Luo, P., Neches, R. (1992): Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. CHI '92 Conference Proc.
- Uhr, H. (2003): TOMBOLA: Simulation and User-Specific Presentation of Executable Task Models. HCI International 2003, Kreta.
- Weaver, W. (1949): Recent contributions to the mathematical theory of communication. In C. Shannon & W. Weaver (Eds.), *The Mathematical Theory of Communication*; Urbana, USA, University of Illinois Press.

- Welie, M. Van, Veer, G.C. van der, and Eliëns, A. (1998): *Euterpe - Tool support for analyzing cooperative environments*: In: Ninth European Conference on Cognitive Ergonomics ,pp. 25-30, August 24-26, 1998, Limerick, Ireland.
- Yantis, S., & Jonides, J. (1990): Abrupt visual onsets and selective attention: voluntary versus automatic allocation. *Journal of Experimental Psychology: Human Perception and Performance*, 16(1), 121— 134.