On autonomous agent modelling for virtual offshore environments

by

©Syed Nasir Danial

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Science

Computational Science Program

Memorial University of Newfoundland

May 2014

St. John's Newfoundland

Abstract

This study explores how simulation based training for offshore emergency situations can embrace software agents who exhibit human-like behaviour when exposed to hazardous situations such as fire aboard, smoke or explosions. Simulation based training uses a virtual environment to expose a participant to scenarios related to mustering events on offshore oil and gas platforms. These scenarios are also relevant to a number of other industrial applications, as they help rehearse for emergency situations such as installation fires.

The agent model proposed here exploits the concepts of similarity-matching and frequency gambling as the primary knowledge retrieval methods and uses the agent's reliability based selection of appropriate knowledge-units to make a decision in the event of a hazard. The agent's reliability is a probability that it acts rationally, and is estimated as a function of the agent's mental modalities: stress, panic, fear, overconfidence and distraction. The effects of these modalities during simulated harsh weather conditions and hazardous events are presented in the form of computer simulations. These simulations show that the use of the agent-model in a training software would enhance the scope of learning by exposing the human participant to more natural human-like behavior during a simulated hazardous event.

Acknowledgements

All praise be to Allāh (the God) alone, the Sustainer of all the worlds, most Compassionate, ever Merciful, and I send salutations upon His noble prophet Muhammad peace be upon him. After that I would like to admit that neither the present work could be completed nor it could be worth reading, without the help of many people, directly or indirectly, through their written texts or by verbal communications. I would like to thank my supervisor Dr. Faisal Khan for his continued support in the form of guiding me towards important areas to explore and also in the form of correcting my direction from going astray. I would also like to thank Dr. Brian Veitch and Dr. Scott MacKinnon, who were also my supervisors and whose reviews and comments helped me shape my research work, in particular, the articles that evolved as parts of this work. I would also like to thank Dr. Dennis Peters from the Department of Computer Science and Dr. Veeresh Gadag, Professor of Biostatistics from the Faculty of Medicine, for their deep and critical examination of the thesis.

I would like to extend my deepest and sincere thanks to my elders, Muhammad Hanif Shah, my late father — Syed Noor-ul-Haq, my mother, sisters and brothers for their moral support. I know without their love, conviction and prayers I would never be able to complete this work.

To my wife, Bushra, I want to thank wholeheartedly. I consider her patience, and true love as a gift from Almighty God. I thank you very much for your support. Finally,

I would like to thank my children, Ahmed and Yaseen because the time I spent here was actually theirs.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations and Symbols

α	A bit value in calling condition representing fire hazard.
β	Reserved bit in calling condition representing explosion.
γ	Reserved bit in calling condition for burning object.
θ	A symbol representing AP as a *proposition*.
ϕ	Retrieving calling condition, a bit string of length m.
τ_s	The % level of required similarity.
Ψ	The set of all propositions (or equivallently all AP's) retreived in a typical retrieval process by Algorithm 1.
$\omega(.)$	The surprise function.
$\Gamma(x)$	A transformation function, Γ: $x \mapsto y$, where $x \in \mathbb{R}_{\geq 0}$, $y \in [0, 1]$.
ϑ	A biased random number whose value is used as index in the KB for retrieving the required KB-Unit.
ϑ_b	A biased Boolean random number used in modelling slips.
ϵ	The fire hazard neighbourhood — the distance from fire hazard from which the agent can detect the hazard.

`f`	Frequency of bias.
`fh`	Flame height.
`fw`	Flame width.
$\mathtt{p}_{\mathtt{flame}}$	Global flame probability.
`temp`	Flame temperature.
kbu_{ϕ}	The array containing all KB-Units corresponding to ϕ.
h_1, h_2, h_3	Bit values for *hazard level* in the CND.
p	Human reliability probability.
s_1	A bit value for *unidentified object* in the CND.
s_2	A bit value for *ordinary combustible material* in the CND.
s_3	A bit value for *electrical lines* in the CND.
s_4	A bit value for *fuel carrying lines* in the CND.
s_5	A bit value for *metallic dust* in the CND.
s_6	A bit value for *smoke* in the CND.
s_7, s_8, s_9	Bit values for *fire class* in the CND.
$ign(.)$	The ignorance function.
r	The agent reliability index.
`BBD`	Bounding box disorder statistic.

FCR	Flame color rate statistic.
POD	Principle orientation disorder.
CP	Continue with the previous goal.
GM	Go to the muster station.
PA	Pay attention.
UC	Use CO_2 base extinguisher.
UD	Use dry chemical base extinguisher.
UDP	Use dry powder base extinguisher.
UF	Use foam base extinguisher.
UM	Use multipurpose dry chemical extinguisher.
UP	Use purple-K dry chemical base extinguisher.
UW	Use water base extinguisher.
UWC	Use wet chemical base extinguisher.
GOAL_CP	Goal continue with the previous goal, if any.
GOAL_GM	Goal go to the muster station.
GOAL_PA	Goal leave every thing and pay attention.
GOAL_UC	Goal use CO_2 base fire extinguisher.
GOAL_UD	Goal use dry chemical base fire extinguisher.

`GOAL_UDP` Goal use dry poweder base fire extinguisher.

`GOAL_UF` Goal use foam base fire extinguisher.

`GOAL_UM` Goal use multipurpose fire extinguisher.

`GOAL_UP` Goal use purple-K dry chemical base fire extinguisher.

`GOAL_UW` Goal use water base fire extinguisher.

`GOAL_UWC` Goal use wet chemical base fire extinguisher.

`GOAL_FOWLLOW_PATH` A C++ goal class for steering behavior `FollowPath`.

`GOAL_ROUTE` A C++ goal class for switching between goals.

3APL Artificial Autonomous Agents Programming Language.

AF Agent Factor.

AI Artificial Intelligence.

AOP Agent Oriented Programming.

AP Action Phrase.

AVERT Adaptive Virtual Environment for Response Training.

BDI Belief, Desire and Intention.

C++ A computer programming language.

CBR Case-based Reasoning.

CND Calling condition.

GPS	General Problem Solver.
HE	Human Error.
KB	Knowledge base.
KB-Unit	Knowledge base unit.
MAL	The muster alarm.
NASA	The National Aeronautics and Space Administration, USA.
NFPA	National Fire Protection Association, USA.
PAL	The process alarm.
PRS	Procedural Reasoning System.
PSF	Human Performance Shaping Factors.
RCS	Reaction Control System.
Soar	State operator snd result.
VE	Virtual Environment.

Chapter 1

Introduction, Overview and Co-Authorship Statement

1.1 Introduction

In contemporary times, software development has entered a new arena where the application is centered around the concept of autonomous characters, called intelligent agents or simply agents. This particular strand of artificial intelligence has gained tremendous scope and significance both in academic research as well as in industrial applications. The application of agents in real life problems abounds: computer games, intelligent assistants, virtual reality and many more. One such application of intelligent agents is in virtual environments (VEs) that are designed to capture characteristics of harsh environments particularly related to emergency training scenarios pertaining to offshore oil and gas platforms. The present work mainly focuses on a system, called AVERT, that is a VE targeting the training tools for offshore oil & gas operations. The VE is designed in a way that a participant is exposed to various training scenarios and drill exercises. The participant becomes the person who takes control of one of the agents in the VE while the other 'agents' are computer controlled.

The actions that the participant performs are reflected by the agent he/she has embodied, whereas the computer controlled characters are bound with whatever actions their respective computer program allows in a given situation. The computer controlled agents have deterministic behaviour and, though they do possess good steering capabilities, they lack intelligence and autonomous behaviour. The requirement was to model the human behaviour that is natural in any emergency situation so that participants can learn the after-effects of good and bad decisions.

Although there are many platforms and programming languages that, at present, support agent development, modeling human behaviour under extreme conditions has not been given much attention. *Extreme conditions* means high mental stress, panic, fear, overconfidence and distraction that are caused by fire hazards and harsh weather conditions such as heavy rains and winds. Such extreme conditions often reduce one's reliability to act as required. This work provides a means to have an agent model that not only supports intelligent decision making, but also models human fallibility so that a participant may learn the dangers involved in making a particular choice in a given learning scenario. The model defines a feedback mechanism and considers frequency of the past decisions in making a new choice against a given situation. This certainly is a step forward in computer implementation of systems like Case-based reasoning (CBR) in a variety of application areas.

1.2 Research objectives

The research has two main objectives:

1. To construct an intelligent agent model that can be used in VE emergency scenarios for harsh environments such as offshore oil & gas installations.

2. The agent model should consider such cognitive factors that are important in human error development in a typical decision making process.

1.3 Overview

Much of the understanding related to the requirements was developed by formal and informal communications with the author's supervisors and other graduate students in the group. The author's course work also laid a firm foundation in his understanding of the field of human error. The literature review explores a precise scope of the required concepts in cognitive modelling, agent oriented modelling and agent-oriented programming and covers the depth of the related topics in (*i*) basics of agent oriented programming, (*ii*) knowledge representations, (*iii*) knowledge base (KB) design, (*iv*) cognitive psychology, (*v*) human factors study, and (*vi*) related information-theoretical aspects. Also, the bibliographic information regarding literature that is used as in-depth knowledge for modeling and implementation is given at the ends of Chapters 3 & 4.

Knowledge is represented in the KB in the form of knowledge units (KB-Units) retrievable via one or more calling conditions (CNDs) where each KB-Unit embodies a solution to a certain instance of a problem. The problem is to make a decision about whether to leave the environment in case of a fire hazard or to try to extinguish it. In the latter case, the problem becomes deciding what type of fire extinguisher best suits the fire class type involved. Thus, a KB-Unit typically contains the *name* of a fire extinguisher and the CND contains the contextual information in which that fire extinguisher should be employed. The agent model comprises four main algorithms. Algorithm 1 retrieves a set of KB-Units from the agent's KB according to the criteria

of similarity-matching and frequency gambling. This *knowledge retrieval* is followed by a *selection* of one most appropriate KB-Unit in Algorithm 2. The selection of a particular KB-Unit is based on the agent's current knowledge and reliability. Agent reliability is estimated on the basis of human factors considered here. These include: stress, panic, fear, overconfidence-bias, and distraction caused by bad weather conditions. Algorithm 3 is actually a template of *how to use a fire extinguisher*. Algorithm 4 gives the agent feedback from the fire hazard after an extinguisher is applied. This information is used for updating the KB, and thus Algorithm 4 is responsible for making the agent learn from its own actions.

The algorithms described above are implemented as a standalone C++ program. The program is designed by exploiting game AI logic, and important methods and C++ classes are presented in Appendices. Thirteen different test scenarios are created that involve fires as the hazards, alarms as triggers, heavy rains and strong winds as weather conditions that induce distraction, and stress, panic, fear and overconfidence as mental attitudes. Impact of the said factors on agent decisions is estimated. Chapter 3 presents the basic algorithms in knowledge representation, KB design, human factor estimation, agent's reliability estimation, knowledge retrieval and learning through feedback. In Chapter 4, a methodology is proposed to deal with alarms. Two types of alarms are considered here: the process alarm and the muster alarm. The question of how much information the agent has is answered by estimating *surprise* and *ignorance* measures. The latter measure estimates a coarse grained effect of the distribution of probability over a number of KB-Units returned by the retrieval process. Similarly, what does the agent learn from past experience? How and when does it use past knowledge? What is the impact of past knowledge on current decisions? Such questions are treated by experimenting with various scenarios in both Chapters 3 & 4. Chapter 5 is the conclusive chapter that opens with a discussion on

various results and recommends possible future extensions to this work.

1.4 Co-Authorship statement

The research topic was proposed to the author by Dr. Faisal Khan, Dr. Brian Veitch from the Faculty of Engineering and Applied Science and Dr. Scott MacKinnon from the School of Human Kinetics & Recreation. Dr. Khan has contributed to this work by directing the author to the required areas of knowledge pertinent to intelligent agent modeling and reliability assessment. Dr. MacKinnon has provided detailed knowledge in the area of cognitive science and human factors study. Dr. Brian Veitch has contributed by his thorough review of the research papers and in thesis development. In addition, the continuous feedback of the supervisory committee and comments in the development of the agent model and its software implementation was a real contribution towards successful completion of this work.

The author was responsible for composing this thesis. He conducted the literature review, developed the theoretical agent model and its software implementation. He made scenarios to be performed through the software and collected the software results, interpreted them and developed conclusions on the basis of which the recommendations were presented.

Chapter 2

Literature review

2.1 Human error & some disastrous events

Errors happen. Unfortunately, serious errors that occur at very sensitive installations can result in the loss of human lives and wealth. These serious errors point to the importance of previously ignored subjects. Such are the studies of human error, safety sciences, virtual training environments, and in particular, the present study that involves development of an intelligent agent which can be used in offshore virtual training scenarios for emergency evacuation and related matters.

The explosion on Deepwater Horizon (2010), a semi-submersible offshore oil drilling rig, located 60km off the US coast, resulted in the deaths of 11 workers, severe injuries to several others and a massive environmental catastrophe due to the release of 5 million barrels of crude oil (BP.com, 2010). The ensuing accident review identified weaknesses in process-safety measures and deficient risk reduction techniques. The Piper Alpha disaster in 1988 caused 167 fatalities. The GSF Adriatic IV, which was an oil rig located over the Temsah gas production platform in the Egyptian Mediterranean Sea, blew out in 2004. Fortunately, there were no casualties but the platform was damaged beyond repair. The Montara blowout (2009) in Australia resulted in an

oil slick of approximately 180km measured east to west (WAtoday.com.au, 2009). In all these incidents, fire and explosion are considered the main reasons for the losses. However, there are a host of organizational factors that are root causes, such as inconsistent safety measures, failure to identify risks, failure to identify and react to early warnings, failure to protect vulnerable areas, failure to make timely decisions under stressed conditions, lack of communication and lack of appropriate training (Khakzad, Khan, & Amyotte, 2013; Skogdalen, Khorsandi, & Vinnem, 2012; Christou & Myrto, 2012).

2.2 Working definitions of human errors

Reason (1990) gives some working definitions that are serviceable rather than presenting hypothetical ideals.

> Error will be taken as a generic term to encompass all those occasions in which a planned sequence of mental or physical activities fails to achieve its intended outcome, and when these failures cannot be attributed to the intervention of some chance agency.

This involves events when a series of actions do not follow the plan, or when the plan itself is inadequate to achieve the desired outcome. It is also possible that storage and execution both incorporate error. The distinction as to whether the error occurred in the storage phase or in the execution model gives rise to two further definitions:

2.2.1 Slips and lapses

Amaya (2011) says, "Slips are *informed* mistakes". This does not mean that people willingly commit slips, but that though they know what they want to achieve and how to achieve it, somehow they miss something they know during the course of actions. In this sense, slips are not due to lack of knowledge. Reason (1990) says that slips and lapses are "the errors which result from some failure in the execution and/or storage stage of an action sequence, regardless of whether or not the plan which guided them was adequate to achieve its objective."

2.2.2 Mistakes

Reason (1990) defines mistakes as "deficiencies or failures in the judgmental and/or inferential processes involved in the selection of an objective or in the specification of the means to achieve it, irrespective of whether or not the actions directed by this decision-scheme run according to plan". Mistakes are considered to have their origin in thoughtlessness (Amaya, 2011).

It is clear now that *mistakes*, *lapses* and *slips*, which are the primary *error types* (Reason, 1990) occur at the cognitive stages or mechanisms of *planning*, *storage* and *execution* respectively. On the other hand, there exists another kind of error called "error forms" which are recurrent varieties of fallibility that appear in all kinds of cognitive activity, irrespective of error types. They are observed in mistakes, lapses and slips. Their omnipresence in every cognitive activity suggests that they are actually rooted in universal cognitive processes, in particular, the mechanisms that involve knowledge retrieval. Two such errors forms that are particularly important in the

context of the present study are *similarity* and *frequency biases*[1].

2.3 A few cognitive studies in human error

2.3.1 Bartlett's notion of *schema*

The term schema[2] was coined by Barlett to explain systematic errors present in recalling textual data such as a prose passage and pictorial information. Barlett defines a schema as:

> [a]n active organization of past reactions, or of past experiences, which must always be supposed to be operating in any well-adapted organic response. That is, whenever there is any order or regularity of behaviour, a particular response is possible only because it is related to other similar responses which have been serially organized, yet which operate, not simply as individual members coming one after another, but as a unitary mass.

According to Barlett, schemata are mental structures that: (*i*) are active without any effort, attention or awareness, (*ii*) contain past knowledge or experience which is the reason he used *unitary mass* in his definition, and (*iii*) include past knowledge comprising active knowledge structures rather than passive ones. Schemata are high-level structures that contain informational slots or variables for holding information. Each variable is considered to contain only a specific kind of information. If the current input from the world fails to provide the required type of data for given slots, they take on default assignments based on past experience.

[1] See Sections 3.3.3 & 3.3.4 for detail accounts on similarity and frequency biases.
[2] Barlett, 1932, p. 201.

Prominent works that are considered a revival of Barlett's schema theory are Minsky (1974), Rumelhart (1975) and Schmidt (1975). Minsky's main contribution in connection with schema theory was in pattern recognition. He proposed that three dimensional scenes can only effectively be perceived by computers if they are able to anticipate much of what will appear. Rumelhart and Ortony (1977, p. 101) defined schemata as "data structures for representing generic concepts stored in memory". McVee, Dunsmore, and Gavelek (2005) present a brief review of schema theory in regards to literacy studies by reviewing various concepts and examining how recent social and cultural realities may prompt their reconsideration.

2.3.2 Memory: primary & working memories

A few of the empirical studies supervised by Wilhelm Maximilian Wundt (1832-1920) dealt with the psychology of memory. He is known to posterity as the "father of experimental psychology" and the founder of the first psychology laboratory (Kim, 2008). However, Wundt rarely used the term "memory" in his work, and to him memory was a general ability to renew ideas (Scheerer, 1980), possibly due to thinking and recollecting. It is, therefore, hard to establish a real connection of psychology of memory with Wundt. Memory is one of the oldest psychological concepts, described in use since the time of Plato and Aristotle (Danziger, 2001). Although Wundt's contribution of memory as a core psychological concept was not something remarkable (Danziger, 2001), since his era, circa 1905, it has been known that some people possess an *immediate memory span*[3]. This is exactly what Wundt called *span of consciousness* and the *focus of attention*. Now this span of consciousness or focus of attention is recognized as primary memory.

[3]Reason, 1990, p.31.

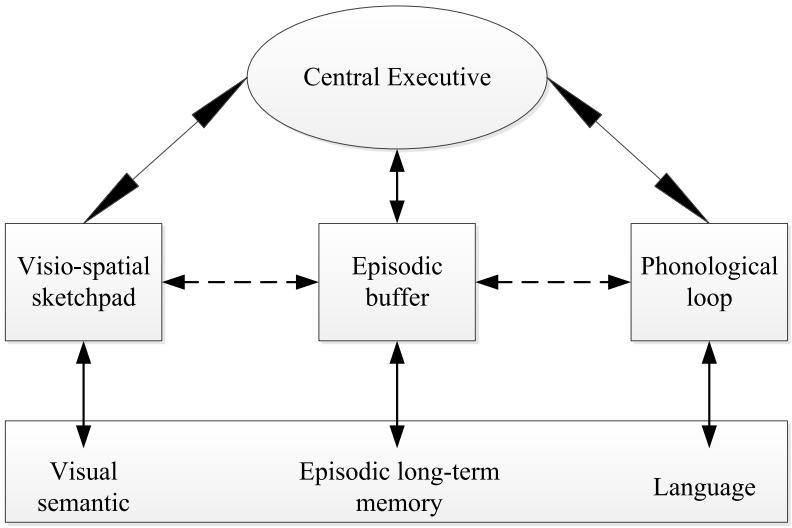

Figure 2.1: Working memory model. (Source: Baddeley (2010), p. 138.).

The notion of *working memory*, evolved from that of short-term memory, was coined by Miller, Galanter, and Pribram in their 1960s classic work *Plans and the structure of behavior*. However, one of the most influential models of working memory was then proposed by Baddeley and Hitch (1974), Baddeley (2007). The model is a multi-component model having an attentional control system by means of the central executive, the visio-spatial sketchpad for holding visual data, the phonological loop to keep verbal-acoustic data and the fourth component, the episodic buffer (Baddeley, 2010). The episodic buffer keeps multidimensional episodes, called *chunks*. These chunks are, at times, used to form an amalgamate of visual and acoustic data, and can add information related with smell and taste (see Figure 2.1).

2.3.3 Newell's General Problem Solver

The General Problem Solver (GPS), which is a rule-base computing model presented by Newell and Simon (1972), has played a vital role in our current understanding

of the pathologies of the human problem solving mechanism. GPS was the first computer program in which the concept of *knowledge* separately dealt with the concept of solution or *a strategy to find a solution*. A *problem space* is defined as consisting of a set of possible states where each mid-level state (including the initial state) serves as a transition state from initial state to the final state. The final knowledge state is supposed to contain the solution of the problem. A problem is posed by giving an initial state of knowledge and then finding a path, by using the heuristic method of *means-end-analysis*, to a final state of knowledge that contains the solution. GPS was implemented in the IPL[4] programming language.

2.3.4 Rasmussen's skill-rule-knowledge framework

This model originated from a verbal protocol study of technicians involved in electronic troubleshooting (Rasmussen & Jensen, 1974). Essentially Rasmussen and Jensen's model deals with the cognitive stage of *performance level* and, more lately, it has become a market standard for systems reliability. Three levels of performance are given below:

2.3.4.1 Skill-base level

At this level, the stored structures of preprogrammed instructions — in a time-space domain — are used to govern human performance. Errors at this level are related to the intrinsic variability of force, space or time coordination (Reason, 1990).

[4]Information Processing Language (IPL) developed by Newell, A., Shaw, C., and Simon, H. A. at RAND Corporation and Carnegie Institute of Technology in 1956.

2.3.4.2 Rule-base level

At this level familiar problems are dealt with by using stored rules in the form of productions of the type `if (condition) then (solution)`. Errors at this level are mainly due to misclassification of conditions or situations that lead to the application of wrong rules.

2.3.4.3 Knowledge-base level

This is the most advanced level in Rasmussen's framework. This level is involved in novel or unexpected situations for which actions must be planned on the fly, using conscious analytical processes and stored knowledge. Errors at this level arise from resource limitations or bounded rationality and incomplete or incorrect knowledge. Rasmussen says that with increasing expertise, the primary focus of control shifts from knowledge-base to skill-base level; however, all three levels can be involved simultaneously in order to solve a given problem.

Lin, Yenn, and Yangb (2010) propose a skill-rule-knowledge base framework that assists in decision making related to the 'types of automation' and 'levels of automation' for the human-automation interaction domain.

2.4 Agent

The notion of agent is now common in AI and also in many disciplines from computer science to economics (Ross, 1973). However, because of the increase in common usage of the notion in diverse ways and in different fields, the meaning of the notion in a particular problem instance remains obscure until a specific account of agenthood

is manifested. In a wider AI-sense, the notion is used for some entity which works continuously and autonomously. Perhaps these two are the most common and unanimously accepted attributes or characteristics of agents for AI researchers who use the notion. Nevertheless, there still remain claims such as, "These are software programs that are, in many respects, alive; and as part of living in this extraordinary network environment, they have begun to manifest many features and actions that were until recently the domain of human beings" (Murch & T. Johnson, 1999). The computing literature suggests that there are some mental qualities which present no sense when ascribed to machines, albeit such an ascription is valid. Consider, for example, the case of a thermostat as explained in McCarthy (1979) that could have the attribution of possessing a mental quality of belief. However, McCarthy maintains that such an ascription, though legitimate, would not aid our understanding of the working of heating systems and thermostats. Moreover, saying that the thermostat "believes" the room is too cold seems to attach much more intricacy to thermostat than what is just because the idiomatic nature of such a sentence is part of our language, not the language we use for such instruments. On the other hand, assigning mental qualities to machines is also advantageous in many domains. To understand this idea, first, consider the case of a flight simulator. The job of a flight simulator is to perform tasks exactly in the same way an aircraft performs, except that the simulator just displays or shows and has no way to move or fly. In this way, a specific combination of commands can be tested to see if this produces a correct sequence of actions or if it dooms the operator to a certain erroneous or undesired output. Yes, it is quite evident that a flight simulator 'simulates' an aircraft, which is a machine, and thus it is manufactured for a high degree of certainty. The situation that an 'agent' often faces is extraordinary because it is frequently targeted to capture some or more of the attributes of the human mind. Second, consider the case of a VE, in which the agents

are designed to behave in as human does. What is this requirement? Is it legitimate? Whether it is legitimate or not it is desirable if people are to be trained for emergency situations that arise in extreme or harsh environments, such as an offshore oil and gas platform. The Emergency situations also arise due to human error. Thus, at least, one mental quality that is responsible for making a human error is fitting to be possessed by the agents in the VE on the grounds of both the legitimacy and desire or need so that a participant can see the agent's actions or decisions that led to a catastrophic situation. Such is the holy grail of the agent, in the context of this study, as this makes the participant learn what was intended.

2.5 Different views about the notion of agent

1. Russel and Subramanian (1995) say, "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors".

2. Wooldridge and Jennings (1995) define the notion by using two separate titles, viz., the weak notion of agency and the strong notion of agency. These forms are defined as:

   (a) "*The Weak Notion of Agency* is a general way to use the term 'agent' to mean hardware or software that should possess the following properties:

      i. *Autonomy*: Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

      ii. *Social ability*: Agents interact with other agents (and possibly humans)

via some kind of agent-communication language;

iii. *Reactivity*: Agents perceive their environment and respond in a timely fashion to changes that occur in it;

iv. *Pro-activeness*: Agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative."

(b) "*The Stronger Notion of Agency*: According to this the notion of 'agent', in addition to having the properties associated with the weak notion of agency, is either conceptualised or implemented using concepts that are more usually applied to humans. For example, it is quite common in AI to characterise an agent using mentalistic notions, such as knowledge, belief, intention, and obligation (Shoham, 1993). Some AI researchers have gone further, and considered emotional agents . . . ".

Such distinction of the notion of agent into weak and strong forms is akin to the goals of the entire field of AI.

3. "An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their common sense counterparts. In this view, therefore, agenthood is in the mind of the programmer: What makes any hardware or software component an agent is precisely the fact that one has chosen to analyze and control it in these mental terms. (Shoham, 1993)".

In other words, Shoham says that the question of "what an agent is" is meaningless because anything, any software or hardware, to which mental qualities

are ascribed, could be termed so. However, whether such an ascription of mental qualities is really legitimate or useful is a matter of thought for discussion.(McCarthy, 1979).

4. "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." (Maes, 1995).

5. "Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions." (Hayes-Roth, 1995).

The attributes of agency are what makes an entity, software or hardware, an agent. Thus the weak notion of agency limits the attributes to four: autonomy, social ability, reactivity and pro-activeness. In the stronger notion of agency, a number of mental qualities such as beliefs, desires or intentions, are required to be ascribed to the agent, and this ascription is considered legitimate when it expresses the same information about the machine that it expresses about a person (McCarthy, 1979).

2.6 Mental qualities and their properties

Shoham (1993) pointed out three modalities: belief, desire, and intention, which were discovered earlier, and counts them as components of an agent's mental state. However, what clear difference the author considers there to be between, for example, desires and choices, is hard to pin point. The author proposes a programming framework for agents, called Agent Oriented Programming (AOP), and expects differ-

ent properties of belief, commitment and capability in different applications of AOP. These mental states are described as temporal and the associated language involves a time component, such as the case where the predicate $holding(robot, cup)^t$ represents the agent's state of holding a cup at time t. Similarly, the author also says that actions, belief, obligation, decision or choice and capability are time dependent qualities. Internal consistency, good faith, introspection and persistence of mental states are some of the most important properties defined by Shoham. The restriction of time on mental qualities looks natural but it also allows the agent to believe in nothing at one time, and shortly afterwards to have belief in every sentence. Thus the AOP framework proposes certain conditions on beliefs, such as having memory and good faith, and only discards a belief if the agent learns a contradictory fact. Similar arguments regarding persistence hold for other mental qualities (Shoham, 1993).

As hinted in the work of Shoham (1993), an important development in modeling agents is the belief, desire and intention (BDI) model later proposed in Rao and M. P. Georgeff (1992, 1995). The BDI logic has its roots in Michael Bratman's theory of human practical reasoning (Bratman, 1987). The theory significantly advances the understanding of agency. According to Velleman (1991) Bratman's theory says that intention should be characterized in terms of its function in rational action — the main role of intention is to make an agent carry out an action. This means that having an intention makes one subject to various ways of acting or reasoning to the limit that one is rational. These ways or norms are reflected as regularities in one's deliberations and actions. Bratman also claims that the functional role of intention is easily seen in plans for the future compared to the immediate intentions. For Bratman, the agent can solve a problem in advance and in parts or stages rather than acting at the time when the action is required. Thus he says,

> I have emphasized the fact that our intentions concerning our future actions are typically elements in larger plans, plans which facilitate coordination both socially and within our own lives, plans which help enable prior deliberation to shape later conduct[5].

and

> Plans, so understood, are intentions writ large. They share the properties of intentions recently noted: they resist reconsideration, and in that sense have inertia; they are conduct controllers, not merely potential conduct influencers; and they provide crucial inputs for further practical reasoning and planning[6].

The example of going to a concert clarifies Bratman's idea of having partial plans. This is an important feature of a plan. Thus a person deciding to go to a concert one night would not immediately settle on a complete plan for the evening. The person might leave till later deliberation about which concert to attend, how to reach the location, with whom to go. As the time of the concert comes closer, the person starts filling out the missing or incomplete knowledge in the plan. In this way, an initially partial plan becomes a complete plan and the deliberation is observed in stages. A second important feature of a plan is its hierarchical nature. Plans concerning ends embed plans concerning means and preliminary steps. Thus if a person decides to go to a concert, he could imagine listening to his favourite singer. Bratman says that agent's rationality can be determined by its intentions. For example, if an agent has an intention, I, at time t_1, and that the agent retains I until some later time t_2 without reconsidering, then the agent seems rational during the time t_1 to t_2 in intending I. McCann (1991) says that according to Bratman's theory intentions serve

[5]Bratman, 1987, chapter 3, p. 28.
[6]Bratman, 1987, chapter 3, p. 29.

as the foundation for plans, and plans are to be filled out by proper substance in a timely manner.

2.7 Agent-oriented modelling approaches

An overwhelming number of programming languages, frameworks and platforms exist that support agent-oriented modelling of a system or process. A few of the most important in the context of the present study are discussed here. The interested reader can refer to Sterling and Taveter (2009)and Murch and T. Johnson (1999) for an overview of the field and the current trends in regards to applications. A good treatise on chunking — a procedure to generate *chunks* of information from experience and store them in memory in a way that they could be retrieved in similar future situations — as a learning mechanism in Soar, discussed in Section 2.7.2, is reported in Schalkoff (2011). For computational aspects and mathematical rigour see Agre and Rosenschein (2006), especially the treatment of the agent by Beer (1995)[7] in terms of dynamical systems theory, where the agent states are shown to be represented as a continuous-time dynamical system. Furthermore, Beer also gives some simple examples revealing a beautiful commonality in how an iterative map, a finite state machine and a differential equation, seemingly different structures, could be used to model a state-space of a system and thereby an agent. Now, before proceeding further it should be kept in mind that any rule based system that can perceive and act in an environment can be considered as an agent though in terms of the weak notion of agency (Russell & Norvig, 1995). Adding sophisticated algorithms to learn and make intelligent rational decisions strengthens an agent. Adding mental attitudes should be considered one step further towards the notion of the strong form of agency.

[7]This paper also appeared in Agre and Rosenschein (2006).

2.7.1 The BDI agent model

A BDI agent, as a software model (Rao & M. P. Georgeff, 1992, 1995), is an abstract architecture that stands out as having a large influence on agent based modelling and development in the past two decades. The popularity of this model may be attributed to the presence of mental attitudes that make a machine, either software or hardware, resemble some modalities of a human mind. As originally presented, it is primarily an abstract architecture and not precisely defined (Sterling & Taveter, 2009). Section 2.8 describes some popular systems that are based on the BDI agent model. In general, a BDI agent has a library of plans and uses different data structures to keep the agent's beliefs, desires and intentions. The agent, upon interaction with an event, picks up and executes a plan for which the invocation conditions and preconditions are satisfied in the event. Here, beliefs refer to the informational state of the agent about itself and its environment. The beliefs, stored in a rule-base, thus may lead to the construction of other beliefs. The veracity of a belief is, nevertheless, not necessarily established at the time of its creation. The agent's desires are motivational factors. Goals are special types of desires which bring the agent into active pursuit to fulfill them. Intentions are the deliberative states of the agent and to some extent are the committed desires. As Bratman proposes that intentions should be understood by means of plans, the software model implements intentions in the form of *plans* which are sets of concrete or atomic actions — actions that can be performed without using the definitions of other actions — the agent can perform in order to achieve or fulfill what the plans intend. Nonetheless, as discussed in Phung, Winikoff, and Padgham (2005), a BDI agent does not learn from its past experience, though it can adapt to changing environment but the model lacks a clear learning mechanism. Similarly, the BDI model does not define an explicit formalism to work in a multi-agent framework (M. Georgeff, Pell, Pollack,

Tambe, & Wooldridge, 1999). Though, not including the BDI logic, the present agent model defines a clear learning mechanism by exploiting the agent's past experience that is maintained with the help of frequency bias.

2.7.2 The Soar framework

The Soar framework (Laird, Rosenbloom, & Newell, 1986, 1987) is an agent development platform. It aims at modelling *general intelligence* — a characteristic feature of human beings — by exploiting a *unified theory of cognition*[8](Newell, 1990) so that the agent is able to enjoy the full range of capabilities possible to an intelligent agent. Historically, the name Soar was an acronym for State Operator And Result. Due to its vast scope, the project is a long-term, multidisciplinary effort. The interested reader is referred to the works of Newell (1990, 1992a, 1992b) and Rosenbloom, Laird, and Newell (1992) for an account of early developments.

2.7.3 The Case-base Reasoning agent model

The Case-base Reasoning (CBR) systems (Aamodt & Plaza, 1994) rely on past similar experiences retrieved from memory, preferably by means of a reconstructive memory search (Kolodner, 1983), to make current decisions. Yang (2013) develops a cloud energy-saving and CBR information agent for the internet that can explore related technologies in order to establish a web service platform. CBR is successful in areas that require experience based problem solving (Bergmann et al., 2003). Aamodt and Plaza (1994) report a four step decision making process, in which, at first, the agent *retrieves* from memory cases that are like the one under consideration. Then it *re-uses*

[8]Another equally important development in terms of general intelligence is the ACT-R cognitive architecture. See (J. R. Anderson et al., 2004; Taatgen, Lebiere, & Anderson, 2006).

by first mapping the old solution into the current scenario, that may involve *adapting* into the new scenario. The third step, the *revise* step, is to test the old solution in the new scenario, say, by means of simulating and testing the results for desirability. The last step involves *retaining* the adapted solution in memory as a new case.

2.8 Applications

2.8.1 The Procedural Reasoning System model

The Procedural Reasoning System (PRS) (M. P. Georgeff & Lansky, 1986; Ingrand, Georgeff, & Rao, 1992) has five major components as depicted in Figure 2.2 and is considered to follow much of the theoretical foundation of the BDI agent model. The first component of the PRS is a rule base about the knowledge of the world the agent is inhabiting. The second component is the set of goals, which plays the role of desires the agent aims to achieve. The goals that the agent is currently trying to achieve form the set of its intentions. This set of intentions is, therefore, the third component of the PRS system. The fourth component is the library of plans. The fifth is the interpreter which coordinates the entire agent behaviour including sensing, acting, updating the beliefs, and choosing a plan to invoke (Sterling & Taveter, 2009).

The BDI-interpreter as described by Rao and M. P. Georgeff is abstract, as it leaves many things to the system implementer. Most BDI systems take an event processing approach and the updates to the agent's beliefs are indicated by events. Similarly, the BDI architecture given in Figure 2.2 is based on a single agent view. A multi-agent system would, therefore, require coordination among individual agent's interpreters. The PRS agent was developed in Lisp and was used on the NASA space shuttle (M. P.

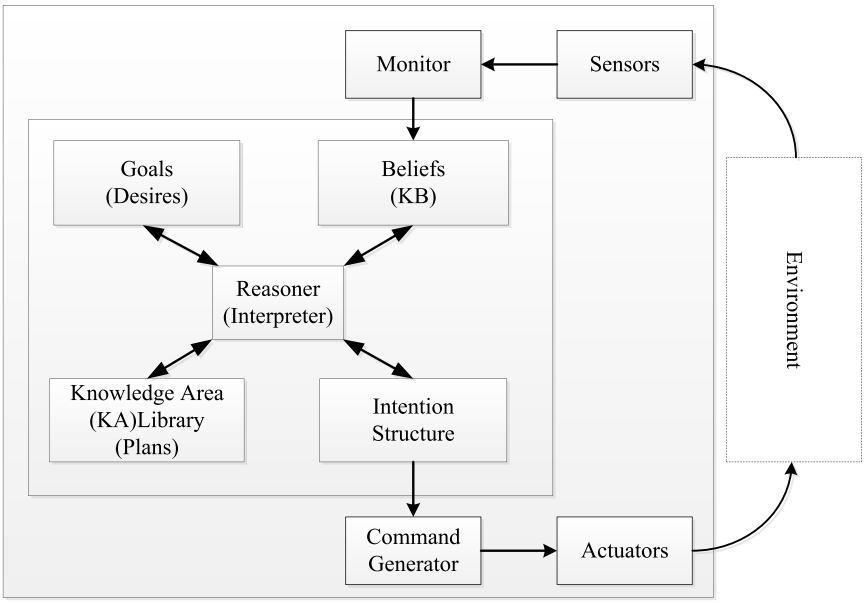

Figure 2.2: The BDI agent model implemented in PRS. (Source: Sterling and Taveter (2009), p. 146.).

Georgeff & Ingrand, 1990) where it was installed as a monitoring and fault detection system for the Reaction Control System (RCS). The system maintained over 1000 facts about RCS, including over 650 facts about the forward RCS alone, half of which are updated continuously.

2.8.2 The AgentSpeak & the Jason platform

In 1996 Rao presented a seminal paper in which he spoke about an abstract programming language, AgentSpeak (Rao, 1996), for his BDI agents. The main language constructs are *beliefs*, *goals*, and *plans* and the architecture supports a *belief base*, a *plan library*, an *event set*, and an *intention set* which are defined in the same way as originally proposed in the BDI model. As a language, AgentSpeak offers an elegant

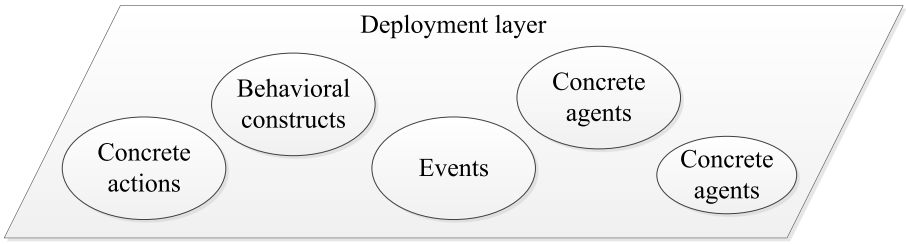

Figure 2.3: The Jason Conceptual Space at deployment layer (Source: Sterling and Taveter (2009), p. 42.).

syntax similar to any other logic programming language in which beliefs are stored in the form of predicates and plans have their own syntax.

Jason (Bordini, Hübner, & Wooldridge, 2007) is an open source, multi-agent system development platform that acts as an interpreter for an extended version of the AgentSpeak. Jason is developed in Java and is supported in JEdit. Jason agents are allowed to be distributed over the internet. They have a belief base and a plan library. A goal expresses the agent's need to have a certain state be true in the environment. Jason associates all the plans with goals.

There are two types of *triggering events* the Jason supports. The conceptual space of the Jason environment (see Figure 2.3) at the deployment layer contains *concrete agents* and *concrete objects*, which are entities having concrete manifestation in the environment. The Jason interpreter updates its events in each execution cycle. Then it unifies each event with *triggering events* in the heads of the plans and generates all *relevant plans* for the *triggering event*. Now the context part of the plan is checked to decide if it follows from the agent's beliefs, and this determines the set of applicable plans. Finally, the interpreter chooses a single applicable plan from the set and executes it.

2.8.3 3APL

The 3APL[9](Dastani, van Riemsdijk, & Meyer, 2005) is an agent programming language based on *modal agent logics*. A 3APL agent has beliefs, plans, goals and reasoning rules and is composed of a belief base, a goal base, an action base, and two rule bases, one for goal planning rules and the other for revision planning rules. The beliefs of a 3APL agent are stored in terms of Prolog facts and rules. Conceptually, beliefs make a set of concrete objects and relationships within them. Similarly a goal is a behavioural construct, and the goal base is a set of goals, each implemented as a conjunction of ground Prolog atoms. 3APL also supports *program operators* which are behavioural constructs and are used for composing plans from basic actions.

The *basic actions* are the concrete actions of the conceptual space and they can take the form of any of five types: (*i*) *mental actions*, which result as a change in the agent's belief base, if successfully executed, (*ii*) *communicative actions*, which are used to pass messages to other agents, (*iii*) *external actions*, which are used to change the external environment in which the agent is operated, (*iv*) *test actions*, which check if a logical derivation is possible from the agent's belief base, and, (*v*) *abstract plans*, which are abstract representations of a plan that can be instantiated with a more concrete plan during execution.

2.8.4 The JACK® intelligent agent

P. Busetta, Howden, Rönnquist, and Hodgson (2000) report that developing a module for BDI requires four important aspects to be well defined. The first step is to define beliefs and their visibility outside the module. The second step involves identification

[9]See Sterling and Taveter (2009) for a detailed account and sample programs of 3APL.

of relevant events and their visibility, because agent development frameworks, such as the JACK Intelligent Agent, typically offer an event-driven programming environment. At the third step the designer has to identify algorithms that best suit event processing. The fourth and the last step is to develop plans that work within the module. The authors introduce a concept of capability which acts as a cluster of beliefs, plans, events and scoping rules over the agents and present an effective way to group and combine the mental attitudes and control the propagation of beliefs and events.

JACK® Intelligent Agent (Paolo Busetta, Ronnquist, Hodgson, & Lucas, 1999) was originally built by Agent Oriented Software Pty. Ltd. (AOS[10]), of Melbourne, Australia, in 1997. It is a framework for multi-agent system development, designed on top of JAVA™programming language and is one of the commercially stable frameworks available today. JACK® agent is defined in terms of the BDI model. Some of its features are given here:

syntactical additions: this includes *keywords* to identify the main components of an agent, a set of statements to declare agent attributes such as the information for beliefs or for an event, static relationships such as which plans can be adopted to react to a certain event, a set of statements for the manipulation of the agent's state.

using JAVA™*statements:* the developer can use JAVA™statements within the agent components.

logical variables: the developer can declare and use logical variables, in particular, to query the agent's state.

JACK® is a commercially available product from AOS and its variant CoJACK™models

[10]http://aosgrp.com/

human cognition, including the effects of moderators, such as fatigue and fear, on human performance.

Chapter 3

Dynamic learning and adapting

An AI based approach to enhance virtual training

Authors: Syed Nasir Danial, Faisal Khan, Brian Veitch, and Scott MacKinnon.

Submitted to: Cognitive Systems Research

Date of Submission: September 18, 2013.

Simulation based training for offshore emergency situations would benefit from scenarios that include interactions with software agents who exhibit human like behaviour when exposed to hazardous situations such as fire on-board, smoke or explosions. Goal based agents reacting to a dynamic environment enhance training scenarios provided they have a speedy system to think and decide a course of action under a given situation. We present a model that can integrate in hierarchical goal-based agent architectures and bring a more natural thinking process in which human factors are considered as key elements in the decision making process.

Keywords: cognitive modelling, human error estimation, similarity-matching, frequency bias, agent for virtual environment, goal driven agent.

3.1 Introduction

Agent oriented modeling of real life problems is now becoming a standard practice in the software industry, notwithstanding the notion that an 'agent' often lacks a clear and unanimous understanding with respect to attributes of agency (Wooldridge & Jennings, 1995). Nevertheless, agent oriented modeling has gained tremendous scope and interest in academic research as well as in industry. An important development is the BDI model (Rao & Georgeff, 1995), according to which an agent has beliefs about the information (about the world including itself), desires as the motivation, and intentions as deliberative states; it decides moment by moment about what to do in order to achieve a goal or fulfill a desire. Lincoln and Veres (2013) developed a natural language design environment, called *sEnglish*, for programming complex robotic agent systems based on BDI architecture. The system provides easy access to agent's shared knowledge regarding operational logic and skill execution related matters to human operators so as to facilitate prototyping of physical agent systems in various simulations or hardware. In (Russel & Subramanian, 1995) the authors propose a bounded-optimal agent model, where bounded-optimality can be seen as a realistically rational behaviour of the agent subject to available computational resources. A method that uses a semantic similarity approach within the framework of CBR to facilitate the use of contradiction matrix is proposed in (Yan, Zanni-Merk, Rousselot, & Cavallucci, 2013). Alterman (1988) uses CBR to reuse old or pre-stored plans in novel situations. The knowledgebase is stored in the form of a network and a plan's position in the network determines the background knowledge of a prestored plan. The network contains category, partonomic, causal and role knowledge and the category knowledge is used for retrieving the prestored plans by means of similarity-matching.

The mental attitudes: beliefs, desires and intentions, though good enough to model a variety of real life problems in various domains, seem unrealistic when it comes to modeling an agent bounded by human error and situated in a harsh environment. The modeling of such an agent requires recalling the laws of association of memory. "Whatever appears in the mind must be introduced; and, when introduced, it is as the associate of something already there. This is as true of what you are recollecting as it is of everything else you think of" (James, 1908). In other words, the human mind searches through memory for desired knowledge by matching stored CNDs with that of the one present in a question in the form of cues. In case of insufficient specification of a retrieval cue, a number of partially matched knowledge units can be activated at various stages of the search. This concept, called similarity-matching, has a wide acceptance in cognitive science (Reason, 1990). An often used knowledge unit has a higher activation level than the one less frequently employed. Thus, on the one hand, the search process generates a number of knowledge units by employing similarity-matching, on the other hand, the knowledge unit which has frequently been used in the past is favoured this time too. In short, the knowledge retrieval at any moment is biased towards the high frequency candidate — frequency gambling (Reason, 1990).

This article presents an agent model suitable for use in a VE to expose a participant to scenarios related to mustering events on offshore oil and gas platforms. The scenarios used in this article are *mental simulations* in the sense of Klein (1998) and these simulations are designed to test the model for interesting situations involving fire hazards. The purpose of these simulations is to show what decision the agent takes given a situation involving a limited number of parameters that form an emerging hazard. At present, the agent considers only fire hazards of low and high intensities. In the former case, it decides *what type* of fire extinguisher is best to extinguish the fire and in the latter case it should decide to *escape* to the muster station. The agent

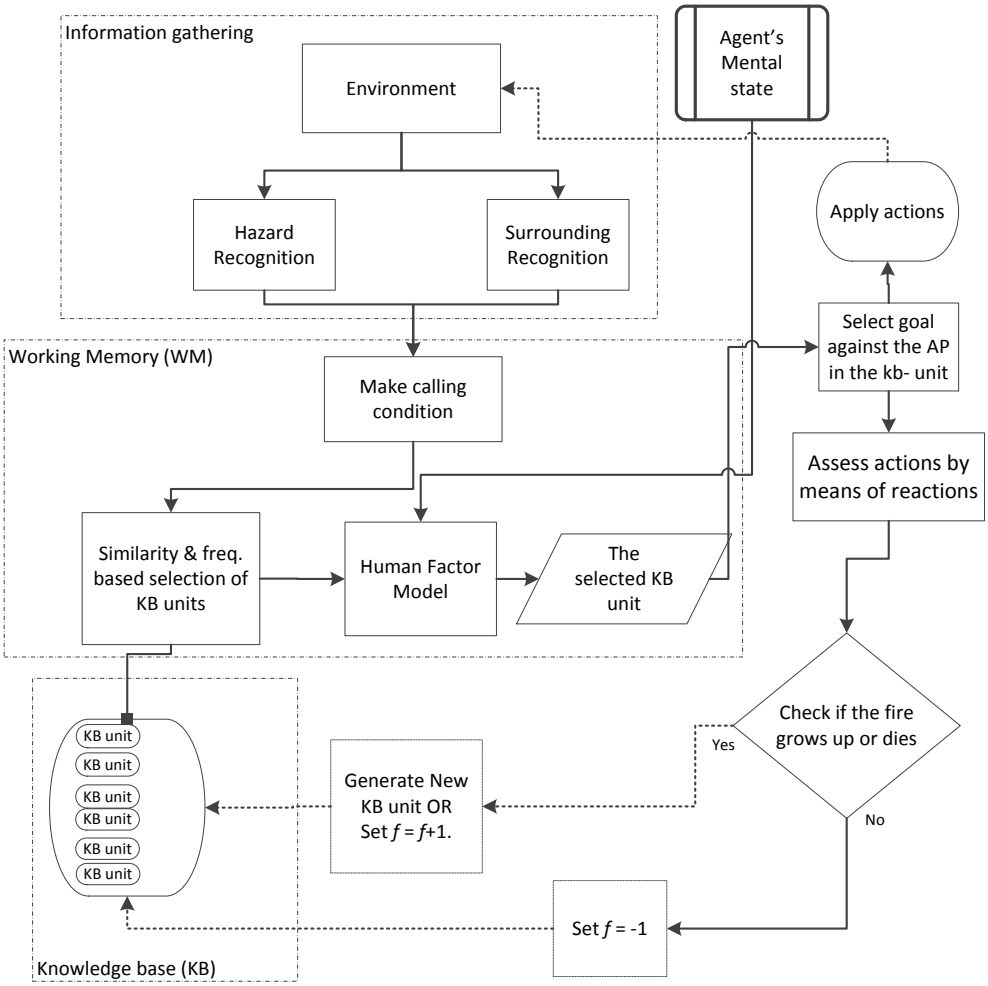

Figure 3.1: The agent model.

uses similarity-matching as a search process guided by frequency of bias to retrieve a set of possible solutions. This retrieval is followed by a selection of a single solution on the basis of the agent's current mental state, which is a reflection of its level of stress. A solution is termed here as an action-phrase (AP), which is a binary-word stored in the agent's KB. The proposed agent model is relevant to a number of industrial applications where the corresponding VEs model emergency situations such as installation fires. The need for high fidelity VEs for training purposes becomes obvious due to a number of industrial accidents. The April 20, 2010 explosion on Deepwater Horizon, located 60km offshore the US coast, resulted in the deaths of 11

Table 3.1: Fire classes with reactions due to different extinguishers. A '+' symbol means the extinguisher's suitability and a '−' symbol means the opposite.

Fire extinguisher	Fire classes				
	A	B	C	D	K
Water base	+	−	−	−	−
Foam base	+	+	−	−	−
CO_2 base	−	+	+	−	−
Dry chemical (sodium bicarbonate) base	−	+	+	−	−
Purple K dry chemical (potassium bicarbonate) base	−	+	+	−	−
Multipurpose dry chemical (ammonium phosphate) base	+	+	+	−	−
Dry powder base	−	−	−	+	−
Wet chemical (potassium acetate) base	−	−	−	−	+

workers, severe injuries to several others and a massive environmental catastrophe due to the release of 5 million barrels of crude oil (BP.com, 2010). The ensuing accident review identified weaknesses in process-safety measures and deficient risk reduction techniques. Other disasters include: Piper Alpha explosion (1988) which caused 167 fatalities due to explosions and fire on board; Adriatic IV blowout (2004), while there were no casualties, the platform was damaged beyond repair; and the Montara blowout (2009) in Australia which resulted an oil slick of approximately 180km measured East to West (WAtoday.com.au, 2009). Fire and explosion are considered the main reasons for the losses. However, there are a host of organizational factors which are root causes, such as inconsistent safety measures, failure to identify risks, failure to identify and react to early warnings, failure to protect vulnerable areas, failure to make timely decisions under stressed conditions, lack of communication and lack of appropriate training (Khakzad, Khan, & Amyotte, 2013; Skogdalen, Khorsandi, & Vinnem, 2012; Christou & Myrto, 2012). In the event of a hazard, safe evacuation of people is one of the primary concerns. However, people need to be trained for dealing with small, controllable fires because such fires have potential to serve as the source of ignition for larger events and could result in compromising hundreds of lives.

Table 3.2: A portion of KB showing KB-Units associated with CNDs.

CND	KB-Unit	
	AP	f
100100010000000	UW	0
100100010000000	UC	−1
100100000100010	UC	0
100100001000110	UW	−1
100110000000001	GM	0
...	...	...

Sometimes accident scenarios are focused on technical and engineering design related aspects of a plant, such as, leakage from a pipeline or failure of a sprinkler system, so as to assess the involved risk (Khan, Sadiq, & Hussain, 2002). It is equally important to model accident scenarios where human reactions could influence the outcome of an event. Clearly, this involves a dynamic VE and the agent should make decisions in such a way not to violate human reliability criteria. That is, the decision making ability should consider human fallibility under extremely stressed conditions, rather than just random selection of solutions from memory.

3.2 Fire hazards types and the AI based scenario analysis

A hazard is classified as either controllable or uncontrollable (or equivalently low or high hazards) in the context of the present study. In case of a low hazard, the selection of an appropriate fire extinguishing agent is essential in fire fighting (see Table 3.1) as described by the US-National Fire Protection Association (NFPA)-10 standards (NFPA.org, 2013). NFPA classifies fires as: A, B, C, D, and K. The class A fire is due to burning of ordinary combustibles such as wood, paper, cloth, rubber or

plastic. The class B fire is caused by igniting flammable liquids. The class C fire is due to energized electrical equipments. The class D fire is caused by burning of certain metals and the class K fire is due to burning of cooking medium.

When the agent observes and recognizes a fire and its surrounding objects, then immediately it generates a CND. The CND is used to search the KB in order to find associated knowledge or KB-Units that contain information about what to do in the current situation. If there is perfect match or matches in the KB and the agent is not under stressors (see Section 3.3) then a KB-Unit is selected based on reliability against which a goal is identified. Once a goal is selected, the agent executes its specific actions, likely following standard operating procedures.

3.3 Methodology

3.3.1 Knowledge representation

We refer the interested reader to Sloman (1995) and Anderson and McCartney (2003) for a discussion on different forms of knowledge representation and making inferences over such forms, such as diagrams or maps. Though much of AI research deals in logic or logic based representation of knowledge, here knowledge is represented in the form of an associative database (Kruetzer & McKenzie, 1991) that also allows duplicate elements. Primarily, the KB comprises a number of KB-Units associated with and retrievable via CNDs. Table 3.2 shows some of the KB-Units and associated CNDs that are used in this work.

Table 3.3: List of APs with associated goal classes.

No.	AP	Goals / goal classes
1.	UW	GOAL_UW
2.	UF	GOAL_UF
3.	UC	GOAL_UC
4.	UD	GOAL_UD
5.	UP	GOAL_UP
6.	UM	GOAL_UM
7.	UDP	GOAL_UDP
8.	UWC	GOAL_UWC
9.	GM	GOAL_GM

3.3.2 The knowledgebase

The agent's KB is a repository of a potentially large number of KB-Units. The KB-Units have information about what to do in a given emergency situation arising from a fire hazard. This includes whether to use a specific fire extinguisher in a particular scenario or to remove oneself from dangers posed by the hazard. The information contained in a KB-Unit needs to be transformed into a set of actions (or a plan) so as to act in response to a situation.

3.3.2.1 The calling condition

A CND represents the context information regarding a decision. The context includes surrounding conditions when a hazard happens and possibly the type of fuel involved. It is, therefore, treated as a key in the KB. Each of its constituents stores a binary value, where a '1' represents the presence and a '0' absence of a certain object or event in the environment. We represent it as a bit string:$\alpha\beta\gamma h_1 h_2 h_3 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9$. The Greek letters are used to denote the hazards; h_1, h_2 and h_3 collectively are for hazard level and $s_1, s_2, \ldots s_6$ represent objects in the surroundings where the hazard

Algorithm 1: Similarity based Knowledge Retrieval.

Input: Retrieving CND ϕ, and the percent level of required similarity τ_s. ϕ is a bit string of length m.

Output: The array kbu_ϕ containing all KB-Units corresponding to ϕ.

$q = 0$;
for *I=1 to size of KB* **do**
    `// I is the index that runs in KB`
    h=hamming distance of ϕ and ϕ_I ; `//` ϕ_I `is the CND of` I^{th} `KB-Unit`
    sim=m-h ; `//` sim `is the number of similar bits`
    if $sim \times 100/m >= \tau_s$ **then**
        $q = q + 1$;
        $kbu_\phi[q]$ = retrieve from KB the KB-Unit associated with ϕ_I such that whenever $\phi = \phi_I$ it is the case that $\mathtt{f}_{\phi_I} \geq 0$, where $\mathtt{f}_{\phi_I}$ is the frequency value in the KB-Unit corresponding to ϕ_I;
        $simkeys[q]$=sim;
    end
end
`sort` kbu_ϕ `// primary sort keys:` $simkeys$`, and secondary sort keys are f values stored within each KB-Unit`
`move` all elements of kbu_ϕ having negative `f` in the last;
`find` and `remove` duplicates starting from last element to the first;

is detected. The bits s_7, s_8 and s_9 are reserved to store the fire class information, so these bits are used in those cases when the agent has information about the type of fuel involved. This work focuses on fire hazards, the presence of which sets α (leaving β and γ unused now). The surrounding objects are ordinary combustible elements (s_2), electrical boards/lines (s_3), fuel carrying lines (s_4), metallic dust (s_5) and smoke (s_6). The bit s_1 is set when an unidentified object is found.

3.3.2.2 The knowledge unit

A KB-Unit is a data structure to hold an AP and its frequency (`f`). An AP has one of the following values: 'use water base extinguisher' (`UW`), 'use foam base extinguisher' (`UF`), 'use CO_2 base extinguisher' (`UC`), 'use dry chemical base extinguisher'

Algorithm 2: Agent Reliability based Selection of AP.

Input: Output of Algorithm 1: kbu_{ϕ} of size q, and agent reliability p.
Output: The action-phrase: $a_{\phi p}$.
$pr[1] = p$;
for I=*1 to* $q-1$ **do**
    $pr[I+1]$=$p \times (1-p)^I$;
end
$pr[q]$=$(1-p)^{q-1}$;
`construct` a biased random number generator **brand** by using pr as weights as explained in Section 3.3.5;
ϑ= `call` **brand**;
$a_{\phi p}$= AP in kbu_{ϕ} $[n]$;

(`UD`), 'use purple K dry chemical base extinguisher' (`UP`), 'use multipurpose dry chemical extinguisher' (`UM`), 'use dry powder base extinguisher' (`UDP`), 'use wet chemical base extinguisher' (`UWC`) and 'go to muster station' (`GM`). Each of these APs has an associated goal which contains specific logic or a plan to achieve what is intended in that AP. Table 3.3 shows a list of goals against the related AP used in this work.

At the start of the scenario, all `f`s associated with KB-Units are set to either '0' or '-1', with 0 meaning that the AP is good to be used with the associated CND and also that in the past the agent has never used this KB-Unit. The value of `f` $= -1$ means that the use of AP (of the same KB-Unit) is a wrong choice in the case referred to by its associated CND. The KB associates a KB-Unit with one or more CNDs or vice-versa. In this way, the CND serves as the contextual information, as said earlier, about when or in what circumstances the (associated) KB-Unit should be used.

3.3.3 Similarity-matching

Similarity matching has a fundamental role in the theories of knowledge and behavior. It is considered a primary basis of memory search (Tversky, 1977). In fact, the concept

is ubiquitous in psychology as it appears across a wide variety of memory theories. As explained in (Reason, 1990), a typical general knowledge question delivers a set of retrieval cues to long-term memory, i.e., KB. The cues delivered this way, activate, in an automatic fashion, the stored items that possess attributes that match either wholly or partially with the CNDs in the question.

The concept has a wide application in computer science in fields like computer vision and multimedia. Majumdar and Samanta (2013) introduced a novel similarity measure based on Hausdorff and set theoretical metrics for vague soft sets. The authors presented how these similarity measures could be used in a decision-making problem. Smeulders, Worring, Santini, Gupta, and Jain (2000) reviewed a number of papers in content-based retrieval from images. Such similarity functions are defined to rank different types of similarity, such as similarity between features (e.g. color), similarity of salient features, similarity of object silhouettes, similarity of structural features and the similarity at a semantic level. Defining similarity measures to perform image alignment is discussed in Brooks, Arbel, and Precup (2008). Stentiford (2007) defines a measure of similarity that does not require prior specification of features and based on only very weak assumptions on the nature of the features in a recognition process. We refer to Santini and Jain (1996) for a discussion on various similarity measures defined for metric and non-metric spaces. Some important models which assume that similarity between two objects can be measured by using a distance function in metric space are Euclidean distance model, City block model and Thurstone-Shepard model. The Feature Contrast Model (Tversky, 1977) is a non-metric set theoretic model for the assessment of similarity between two objects.

Here we adopt a view that similarity-matching should play an important role in searching the knowledge base for decisions which previously were taken in similar situations.

Algorithm 3: Plan or Goal Execution.

Input: Type of fire extinguisher to be used
Output: Arrays of size N to hold: temperature (`temp`), flame width (`fw`) and flame height (`fh`).
Find the location of extinguisher that matches the type.;
Move to the extinguisher's location.;
Carry the extinguisher to a safe distance from fire location.;
Point the extinguisher towards the fire location.;
for *I=1 to N* **do**
    release the extinguishing agent on the fire `// this brings the fire on reaction`;
    temp[I]= get current fire temperature;
    fw[I]= get current flame width;
    fh[I]= get current flame height;
end

The word 'situation' here means the state of the surroundings of the place where a hazard is detected. For example, the agent faces a fire at a place where electricity wirings and panels are very near.

Now this context information is used to make a CND by turning on/off respective bits as explained in Section 3.3.2.1. Since such a CND is generated every time the agent observes a hazard, it is better to call it a retrieving CND. The retrieving CND is matched with each of the stored CNDs of KB one by one. The matching is performed here by calculating the hamming distance function — which is the minimum for an exact match. Algorithm 1 computes all those KB-Units where the ratio of the number of equal bits to the total number of bits is either greater or equal to some arbitrary constant $\tau_s \in [0, 1]$. This way the agent retrieves a set of KB-Units corresponding to a retrieving CND. The agent is set to wander around in the environment in case there are no hazards.

One particular limitation of Algorithm 1 is its exhaustive nature. The only for loop runs through the entire KB and computes the hamming distance — that makes the

complexity of the algorithm O(N) — where N is the size of the KB. Algorithm 1 works on an unordered collection of KB-Units. Considering an ordered collection does not make sense because the key attribute, the retrieving CND, should be similar to the stored CND and that can be true anywhere in the KB unless the KB-Units are stored in classes. The solution to this problem is to make use of rank aggregation (Fagin, Kumar, & Sivakumar, 2003), i.e., to classify the KB by aggregating the similar KB-Units into classes of similar KB-Units and then perform exhaustive search within a class whose rank suggests the presence of the required KB-Unit. This is the problem of similarity-based searching that is solved, in general, by classification (within the KB) and then computing the nearest neighbor or minimal distance (Duch, 2000).

3.3.4 Frequency gambling

The under-specification of a CND due to any reason is the main objective of similarity-matching. Otherwise, in the presence of an exact match the agent uses the corresponding KB-Unit. However, even with exact matches there can be more than one KB-Units found in a typical search. Whether the search ends in KB-Unit having CNDs exactly matched with the retrieving CND, or the KB-Units are found as a result of partial matching, the KB-Unit which has been used most frequently is more likely to be used now. This concept is called frequency-gambling or frequency bias according to which the most frequent KB-Unit has more tendency to come to the output (Reason, 1990). The frequency information is stored within the KB-Unit as a member variable `f` and it is this information that is used here to maintain agent experience. The set of KB-Units found against a retrieving CND, as explained in the Section 3.3.3, is sorted in descending order (see Algorithm 1), first by the number of similar bits and then by their relevant value of `f` so that the data structure holding this set should have the

Algorithm 4: Assess actions and Update KB.

Input: Output from algorithm 3: `temp`, `fw`, `fh`; reference of currently selected kbu_ϕ and CND ϕ.
Output: The KB will be updated.
Fit a linear regression line on `temp`, `fw` and `fh`.
Let m_t, m_w and m_h be the slope of `temp`, `fw` and `fh` respectively.
if $m_t < 0$ *and* $m_w < 0$ *and* $m_h < 0$ **then**
    if *CND corresponding to* $kbu_\phi == \phi$ **then**
        increment `f` in kbu_ϕ;
    end
    else
        add a new KB-Unit in KB with ϕ as the CND and 1 as the value of `f`;
    end
end
else
    if *CND corresponding to* $kbu_\phi == \phi$ **then**
        set `f` in kbu_ϕ with -1;
    end
    else
        add a new KB-Unit in KB with ϕ as the CND and -1 as the value of `f`;
    end
end

most similar and most frequent KB-Units from the onset. Algorithm 1 also pushes KB-Units to the bottom where `f` $= -1$ and removes redundancy by keeping only those that have minimum hamming distance and maximum frequency value.

3.3.5 Human factor estimation

Human factors, as defined by Sullivan (2009), is the study of:

> the dynamics affecting human performance, often in service of the reduction of human error and better systems design. Researchers in this field study the physical, cognitive, psychological, and social contributors to effective task performance, efficient systems use, and user satisfaction.

Table 3.4: Estimates of p for various values of AF.

AF	Estimates of probability, p
[0.00, 0.10]	0.98
(0.10, 0.25]	0.9
(0.25, 0.50]	0.8
(0.50, 0.75]	0.7
(0.75, 1.00]	0.6
(1.00, 1.25]	0.5
(1.25, 1.50]	0.4
(1.50, 1.75]	0.3
(1.75, 2.00]	0.2
(2.00, 2.25]	0.1
(2.25, 2.50]	0.05
(2.50, 2.75]	0.03
(2.75, 3.00]	0.01

The estimation of such factors for harsh environments is normally done in the form of calculating the human reliability, p, which is the probability that a person correctly performs the system-required activities in a required time period (Swain & Guttmann, 1983). Musharraf, Hussain, et al. (2013) reported an analysis of the human performance shaping factors (PSF) during several tasks related to the muster phase on an offshore installation. They showed that the major factors 'to act accordingly' are distraction, stress, action procedure, available time, complexity, fear and training experience. Musharraf, Khan, Veitch, MacKinnon, and Imtiaz (2013) estimated human error probabilities for the offshore emergency environment using Bayesian, Fuzzy and Evidence theory. They used four major factors that influence stress and fatigue, which in turn deteriorates human performance in cold environments. Tompkins (2010) reported that workplace stress may instigate or aggravate a panicked operator response that eventually causes poor performance. The importance of overconfidence in decision-making is well-known. An overconfident person is more likely to focus on those factors that favour his/her decisions and avoid or neglect those that are in contradiction (Reason, 1990).

Here, the purpose of the above discussion is to bring about a way that the agent can mediate these response moderators on a participant's decision-making capabilities. Four factors are selected: stress, panic, fear and overconfidence. A weighted average combines their effect into a single entity termed as 'agent-factor', AF:

$$AF = \frac{w_s \times stress + w_p \times panic + w_f \times fear + w_o \times overconfidence}{w_s + w_p + w_f + w_o}. \tag{3.1}$$

The values of stress, panic, fear and overconfidence are selected from the set: $\{0, 1, 2, 3\}$ with 0 meaning 'no', '1' means 'low', '2' for 'medium' and '3' to represent a 'high' value. The human reliability probability (p) is estimated for the case when all the weights in Equation 3.1 equal unity (see Table 3.4). Algorithm 2 accepts output of Algorithm 1 as its input and receives the reliability p as a second input (see Figure 3.1) to generate as many probability values as the size of the array kbu_ϕ. The first amongst these probability values is the one estimated through Equation 3.1. These probabilities are used to construct a biased random number generator by assigning the probabilities as weights in a discrete distribution and then employing the Mercenne Twister (Matsumoto & Nishimura, 1998), which has a period of $2^{19937} - 1$, to generate a random number. The numbers of the form $2^m - 1$ are called Mersenne numbers (Knuth, 1969) where m is a prime number. Note that every prime number does not give rise to a Mersenne prime. The random number generated this way is used as an index (into the array containing the KB-Units obtained from Algorithm 1) to choose one KB-Unit which appears on the output and which has the AP needed to be used in the current situation.

3.3.6 Learning through experience: the feedback mechanism

The agent learns through experience by observing and assessing the reaction (against its action) from the entity it acted on. The process involves assessment of the reaction in terms of whether the action was right or wrong. As an example, when Algorithm 1 (followed by Algorithm 2) is executed, the agent arrives at a decision. But to this end, it is just a decision in the agent's mind and this has to be translated into a plan or a set of actions that the agent can perform in the VE. As the agent is goal oriented, each AP has an associated goal class, the object of which can be selected in an `if-then-else` statement. The agent also selects an extinguishing tool (see Table 3.1) according to the selected AP and passes this information to the respective goal object. Thus, executing the plan specific to a goal means that the agent starts performing actions according to the goal-specific logic. A typical plan involves steps as shown in Algorithm 3, where the agent acts and receives reaction from the entity it acted on. This reaction is recorded or saved in separate data structures for assessment as depicted in Algorithm 4. Algorithm 4 uses a simple assessment policy to determine whether the fire is extinguished or escalating following the use of the selected extinguishing tool. If the fire is extinguished it means that the actions and thereby the decision which leads to such actions were likely correct and hence the corresponding frequency of bias should be incremented. On the other hand, if the fire is growing the corresponding frequency of bias is set to -1; it would diminish the chances of the selection of the associated KB-Unit the next time the same CND is generated. The case when a KB-Unit is selected due to partial matching of CNDs leads to an addition of a new KB-Unit (see Algorithm 4).

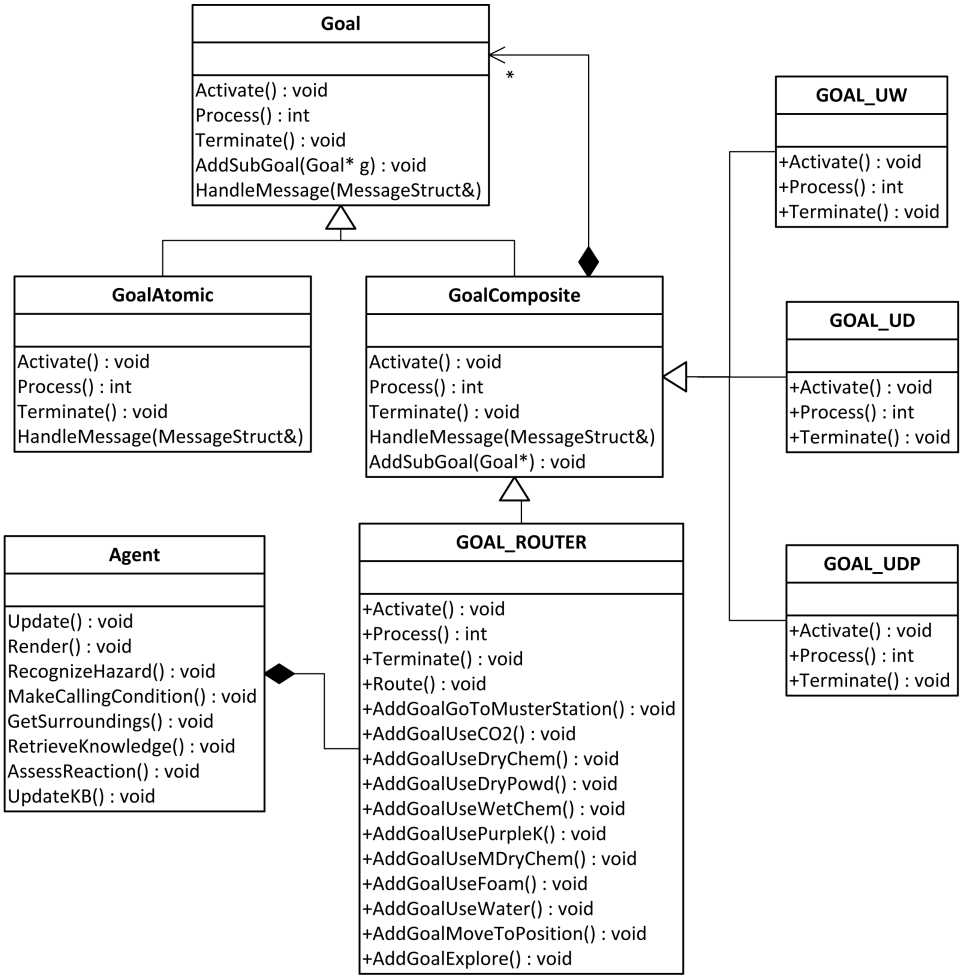

Figure 3.2: Class diagram showing some of the main goal classes.

3.4 Software implementation

The methodology as described in previous sections has been applied to develop a goal-oriented agent in C++ language. The purpose here is to test the agent against various emergency scenarios. The VE is a 2D graphic environment similar to a computer game. It has walls, a muster station, combustible metals, obstacles like chairs, electrical lines, fuel lines, and fire extinguishers installed at different fixed locations. Fire objects update their temperature, flame length and flame width on each update-

cycle of the environment. A navigation graph is used to construct paths from one position to another.

By default, the agent moves all around the VE. As the agent is goal-oriented, each of its steering behaviours (Reynolds, 1999; Buckland, 2005; Millington & Funge, 2009), such as, *arrive*, *seek*, *wander* and *follow-path*, is activated from a respective goal class object. For example, the agent uses *follow-path* behaviour when the object of `GOAL_FOLLOW_PATH` comes in execution. A goal which is not defined in terms of other goals is called 'atomic goal' and a goal that depends on other goals is a 'composite goal'. Thus the goal `GOAL_UW` is a composite goal because its plan needs to execute other goals, such as, `GOAL_MOVE_TO_POSITION` or `GOAL_FOLLOW_PATH` to reach to certain required location on the 2D map. The composite design pattern (Gamma, Helm, Johnson, & Vlissides, 1994) is used to maintain the class hierarchy of the goals. At any moment of time, one of the goals is always active and its plan is in execution. In order to switch between goals, especially when a hazard is seen, the agent compares the selected AP with each goal type and executes the goal accordingly. This process of goal routing is done in `route` method of the class `GOAL_ROUTER`. The `route` method is updated in every update cycle of the agent. Figure 3.2 shows the hierarchy of the goal classes with a few classes for illustration purposes.

The agent detects the fire (i.e. object of `Fire` class). It classifies it in terms of either *no-hazard*, *low-hazard* or *high-hazard* by reading the global flame probability value, $\mathtt{p_{flame}} = (\mathtt{BBD} + \mathtt{POD} + \mathtt{FCR})/3$, (Verstockt et al., 2011), where, `BBD` is the bounding box disorder statistic, `POD` is the principle orientation disorder (Verstockt, Vanoosthuyse, Hoecke, Lambert, & Walle, 2010) and `FCR` is the flame color rate statistic (Chen, Wu, & Chiou, 2004; Verstockt et al., 2011). This $\mathtt{p_{flame}}$ based recognition of fire surely enhances the scope of this work as later on the recognition method could be upgraded

to support image-processing. The obstacles in the vicinity of the fire hazard are noted and a CND is generated. Algorithm 1 is used to find a set of KB-Units relative to the CND and then application of Algorithm 2 selects one KB-Unit from the set. The AP in the selected KB-Unit has an associated goal class (see Table 3.3) and the goal class embodies a plan. Generally, goal classes should have different plans but here `GOAL_UW`, `GOAL_UF`, `GOAL_UC`, `GOAL_UD`, `GOAL_UP`, `GOAL_UM`, `GOAL_UDP`, and `GOAL_UWC` follow the plan as in Algorithm 3 because the way each fire extinguisher should be used is the same in all cases irrespective of the type of the extinguisher. The `GOAL_GM` finds a path to the muster station and allows the agent to follow that path, which eventually brings the agent into the muster station. The plan of `GOAL_GM` and several other atomic goals are not mentioned here for brevity.

As in Figure 3.2, `GOAL_ROUTER` is designed for the purpose of selecting (or making a goal active) a goal against the AP. Once a goal is set to active, its plan is put into action. The assessment and thereby KB update process is done in accordance with Algorithm 4 such that if the goal produces desirable results, such as extinguishing a fire or escape towards muster station, the frequency of the concerned KB-Unit is incremented. Otherwise, in case of undesirable results, the frequency is set to negative unity.

The software is tested by making nine different scenarios. The main parameters which differentiate one scenario from the other are: (*i*) the fire class type, (*ii*) the fire surroundings, (*iii*) the agent's mental state, (*iv*) the agent's learning, and (*v*) the agent's prior experience.The agent's mental state is determined by its stress, panic, fear and over confidence values in Equation 3.1. The reliability values corresponding to different ranges of AF can be seen from Table 3.4.

Table 3.5: The KB rules before simulation, involving class D type fire (not passed to the agent) are: R1 to R8. After simulation: R9 is added into KB. R10 is the rule obtained when the fire class information was sent to the agent. F stands for fire, LH for low hazard, FU, M and E represent fuel lines, electric lines and combustible metals respectively and B and D are fire classes.

Rule	CND	Description	KB-Unit	
			AP	f
R1	100100000100000	F, LH, FU	UM	0
R2	100100000100000	F, LH, FU	UF	0
R3	100100000100000	F, LH, FU	UC	0
R4	100100000100000	F, LH, FU	UD	0
R5	100100000100000	F, LH, FU	UP	0
R6	100100000010000	F, LH, M	UDP	0
R7	100100000100000	F, LH, FU	UW	-1
R8	100100000100000	F, LH, FU	UWC	-1
R9	$c_1 = 100100000110$	F, LH, FU, M	UM	-1
R10	$c_2 = 100100000110110$	F, LH, FU, M, D	UDP	1

3.5 Scenarios and Results

3.5.1 Agent under normal mental state faces class D type fire

The agent is not given the fire class information. Thus the only way to make a decision is by means of surroundings. By observing the fire and its surroundings, the agent generates CND $c_1 = 100100000110$. Since it does not exactly match any of the rules in the KB — meaning that the agent does not know what exactly to do in this situation — it tries to use its previous knowledge. Algorithm 1 returns rules R1 to R8 as shown in Table 3.5. The agent chooses R1, applies a multi-purpose extinguisher to the fire and that escalates the fire level from low to high. The change in the hazard level — showing a failure of the fire extinguisher — is observed by the agent and it searches the KB again. This time it decides to use GM by employing GOAL_GM, because against high level hazards the only strategy is to move to the muster station. The agent also

marks the failure of multi-purpose extinguisher by putting this information in terms of a new rule, R9, into the KB, so as to avoid it in the future. Similarly, keeping all factors constant, we repeat this scenario six times more and discover that in the second, third, fourth and fifth attempts, the agent comes up with UF, UC, UD and UP and uses corresponding goals GOAL_UF, GOAL_UC, GOAL_UD, GOAL_UP respectively. All of these escalate fire and the agent has to abandon the facility by employing GOAL_GM. However, as expected, the agent notes all these wrong attempts in the KB by setting their respective frequencies to -1. In the sixth attempt, the agent comes up with UDP, that is a correct KB-Unit (see Table 3.1) in this scenario, and uses GOAL_UDP to extinguish the fire. This attempt is successful and the agent updates the KB correctly.

We repeat this simulation again but by making slight changes in parameters, such that this time: (*i*) the agent is given the fire class information, and (*ii*) the KB is reset to its original state, i.e., all the updates that had been performed are removed. When the simulation is run, the retreiving CND is $c_2 = 100100000110110$, where the last three bits from the right side, i.e., 110 are now set to represent a D-class fire. The agent searches the KB and does not find any exact matching KB-Unit. The most similar KB-Unit returned by the Algorithm 1 has UDP as the AP with f $= 0$ and is associated with the CND 100100000000110. The agent selects its associated goal, i.e., GOAL_UDP, successfully extinguishes the fire and records a new rule, R10, in the KB. Successive simulations (not reported here) by keeping the surrounding conditions and other factors constant, use R10 as a result of an exact match where each time only frequency is incremented.

The similarity based results as reported above were obtained by a using a similarity criterion of $\geq 80.0\%$. By reducing it to 75% and providing the agent with the fire class information, the agent was able to extract more KB-Units but it used the

same KB-Unit as it used with 80.0% criterion because the Algorithm 2 always assigns probabilities in descending order of the similarity and frequency — with highest probability to the one that has highest similarity rank and highest frequency of past use.

3.5.2 Agent under normal mental state faces fire that is likely to be caused by burning of ordinary combustibles: Default KB is used in each simulation

The agent faces a low hazard fire where the surroundings include only ordinary combustibles. Therefore, the CND does not include information about fire type. The agent has no prior experience about fire fighting, though its KB has relevant knowledge. The simulation is performed 50 times with each time loading the default KB, which equivalently means that the updated KB is not used in successive simulations. This is to test how the agent behaves if it encounters the same scenarios repeatedly. Based on the surrounding information, the agent comes up with three exact matches. These are `UW`, `UF` and `UM`. In 49 out of 50 simulations, the agent selects `UW` that leads to `GOAL_UW` where a water base fire extinguisher is used successfully to extinguish the fire. In 1 simulation, the agent selects `UF` that has the probability of selection as around 0.0196. It employs `GOAL_UF` successfully and updates the KB.

Table 3.6: Results of scenario in the Section 3.5.6.

AP	Occurrences	Outcome
UM	119	success
UF	271	success
UD	8	success
UC	11	success
GM	42	success
UP	4	success
UW then GM	45	failure then success

3.5.3 Agent under normal mental state faces fire that is likely to be caused by burning of ordinary combustibles: Updated KB is used in each simulation

The agent starts with the default KB but in each simulation it updates the KB. The simulation is performed 50 times. Based on the available information, the agent comes up with three exact matches from the KB. These are UW, UF, and UM. In 43 out of 50 simulations, the agent selects UW which leads to GOAL_UW where water base fire extinguisher is deployed successfully. Once it selects UF (having 0.0196 probability) and uses the foam base extinguisher successfully. In the remaining simulations, the agent could not observe the fire in time, because it was wandering in other regions of the VE. Thus when the agent observes the fire, the fire had already turned into a high-hazard which eventually changed the opportunities and the agent decided to go to the muster station instead of trying to extinguish the fire.

3.5.4 Agent under normal mental state faces fire where the surroundings only have fuel lines and electrical lines: Updated KB is used in each simulation

The environment includes a class B fire but this information is not passed to the agent. Thus the decision is based only on the surrounding objects. The simulation is performed 50 times. The agent uses the default KB only in the first simulation. Four exact matches are found which are: `UC`, `UD`, `UP` and `UM`. The agent selected `UC` in 41 simulations, `UD` in 3 simulations and `GM` in 6 cases (due to observing the fire late). The value of `f` is updated each time in respective KB-Units.

3.5.5 Agent under normal mental state faces fire where the surroundings have ordinary combustibles, fuel lines and electrical lines: Updated KB is used in each simulation

The fire class is unknown to the agent so it proceeds by selecting the extinguisher which is as safe as possible. This is done by keeping the information about the surrounding object's type and populating the CND with this information. The ABC-class multi-purpose-dry-chemical extinguishing-agent is chosen, which seems to be the best choice in such a case. The agent selects `UM` 49 times. It is late in observing the fire in one case, so chooses `GM`.

Table 3.7: Results of scenario in the Section 3.5.7.

AP	Occurrences	Outcome
UM	222	success
UC	95	success
UF	542	success
UD	12	success
GM	88	success
UP	30	success
UW then GM	4	failure then success
UDP then GM	4	failure then success
UWC then GM	3	failure then success

3.5.6 Agent under high stress faces fire where the surroundings only have ordinary combustibles and fuel lines

The stress and panic attributes of the agent are set to high and low respectively. The fire class type is B but this information is not passed to the agent. Agent discovers ordinary combustibles and fuel lines near the hazard and generates the CND accordingly. The first simulation uses the default KB and in each of the successive simulations the updated KB from the previous simulation is used. After 500 simulations (see Table 3.6), 45 cases are found to be those where the agent initially makes a wrong decision (by selecting UW) which intensifies the fire and creates a high hazard situation. This new observation makes the agent search the KB again to determine what decision should be made. The KB has only one rule against a high hazard situation and that is to move to the muster station. Thus this time the agent selects GM and goes to the muster station successfully.

Table 3.8: Results of scenario in Section 3.5.8.

AP	Occurrences	Outcome
UM	209	success
UF	523	success
UD	33	success
UC	94	success
GM	116	success
UP	18	success
UW then GM	6	failure then success
UWC then GM	1	failure then success

3.5.7 Agent under high stress faces fire where the surroundings only have ordinary combustibles and fuel lines: Initially trained agent

An agent under a normal mental state is first trained by exposure to several scenarios in which different types of fires with different surroundings were used. The agent made decisions and updated its KB accordingly. Then this agent is put in the environment where a class B type fire hazard is in proximity to ordinary combustible materials and fuel lines. The agent is not given the fire class type. Further the agent is made stressed by setting stress to high and panic to low in the Equation 3.1. The results are shown in Table 3.7.

After 1000 simulations, a total of 11 cases occurred where the agent initially makes a wrong decision (by selecting UW, UDP or UWC) which intensifies the fire and creates a high hazard situation. This new observation, as in the scenario of Section 3.5.6, leads the agent to select GM and hence the agent moves to the muster station successfully. In 88 cases the agent identified the fire too late and created a high hazard situation, so the agent decision of moving to the muster station was correct.

Table 3.9: Results of scenario in Section 3.5.9.

AP	Occurrences	Outcome
UM	193	success
UF	535	success
UD	12	success
UC	22	success
GM	146	success
UP	85	success
UW then GM	5	failure then success
UWC then GM	2	failure then success

3.5.8 Agent under high stress faces fire where the surroundings only have ordinary combustibles and fuel lines: Fire class is known to agent

This scenario is similar to the one presented in Section 3.5.6. The difference is the agent knows that this is a class B fire and each simulation uses the default KB. The KB-Units against exactly matching CNDs are put on high priority as their probability of selection is high compared to less matched and less frequent KB-Units. After 1000 simulations (see Table 3.8) are performed, only 7 cases reported failures and the remaining 993 cases reported successes.

3.5.9 Agent under high stress faces fire where the surroundings only have ordinary combustibles and fuel lines: Fire class is known to agent

This scenario is the same as presented in Section 3.5.8 except that the agent used here is not novice. Before performing these simulations, the agent was trained over several scenarios. Further, each successive simulation uses the KB updated from the last

Table 3.10: Results obtained by CBR toolkit.

Scenario of Section 3.5.2			Scenario of Section 3.5.4		
AP	f	% similarity	AP	f	% similarity
UWC	-1	100	UWC	-1	100
UW	**0**	**100**	UW	-1	100
UP	-1	100	**UP**	**0**	**100**
UM	**0**	**100**	**UM**	**0**	**100**
UF	**0**	**100**	UF	-1	100
UDP	-1	100	UDP	-1	100
UD	-1	100	**UD**	**2**	**100**
UC	-1	100	UD	0	100
UWC	-1	100	**UC**	**1**	**100**
UW	-1	83	UC	0	100
UM	0	83	UWC	-1	83
UF	-1	83	UW	-1	83
UDP	-1	83	UP	-1	83
UM	0	83	UM	0	83
UF	1	83	UF	-1	83

simulation. The agent also knows that this is a class B Fire. After 1000 simulations (see Table 3.9) only 7 cases reported failures and 993 cases reported successes.

3.5.10 Results from case-based reasoning

This section describes the results obtained by employing the scenarios of Section 3.5.2 and 3.5.4 into the CBR architecture. We only wanted to show that our results of similarity-matching, as computed through Algorithm 1, are similar to what could be obtained by other techniques. For this purpose, CBR toolkit in WinProlog from Logic Programming Associates was used. The results are shown in Table 3.10 where the highlighted values represent those KB-Units that were used in respective scenarios.

3.6 Conclusion

This paper discusses a new approach of modeling an intelligent agent that considers the concepts of similarity-matching and frequency bias for knowledge retrieval from a KB where each KB-Unit forms associations with binary CNDs. The final selection of a KB-Unit is made as per agent reliability, that is the probability the agent can perform rationally by retrieving correct or relevant information from the KB. The agent reliability is calculated as per its stress, panic, fear and overconfidence level. The agent model is presented in Figure 3.1 and implemented in a program in which a VE can incorporate fire hazards. A total of 9 scenarios are built and the results are presented.

In conclusion:

1. When the retrieving CND has no perfect match with the stored CNDs in the KB then partially matched KB-Units are identified. The agent in the scenario of Section 3.5.1 has found a single partially matched KB-Unit containing `UDP` when the fire class information is given to the agent. Decreasing the percent similarity level would surely include more APs. The mental state above 'normal' amplifies the chances of committing a wrong decision. Whatever the case, the agent learns and adds a new KB-Unit whose CND would be the same as the retrieving CND. There are two possibilities for the newly added KB-Unit: either it has an AP with $\mathtt{f} = -1$ or $\mathtt{f} = 1$. The latter case is only possible when the agent 'luckily' finds an extinguishing agent that did the job of extinguishing the fire successfully. If no such extinguishing agent is available or is 'known' to the agent, it will mark every other wrong choice with $\mathtt{f} = -1$ and in the subsequent confrontation with the same scenario the search process will put all these KB-

Units having $\mathtt{f} = -1$ to the lowest possible priority or completely ignore all or some of them.

2. Under a normal mental state (scenarios in the Sections 3.5.2 through 3.5.5) the success of the agent is definitive, as long as it has relevant information in the KB, even for cases where the agent does not have prior experience (KB-Units with $\mathtt{f} = 0$). Note that, $\mathtt{f} = -1$ indicates that the AP is a wrong choice under the associated CND.

3. Under stress, a novice agent is found to have 9% failure (see Section 3.5.6). The failure rate reduced to 1.1% when an experienced agent is used under the same conditions (see Section 3.5.7). The failure rate is further reduced in scenarios of the Sections 3.5.8 and 3.5.9 to 0.7% in both scenarios because the information regarding the fuel involved in the fire was also given to the agent and this information narrows the search band.

The evidence from the simulations supports that the agent response based on the current model has a more realistic knowledge retrieval process, and that the decision reasoning is supported by the premises identified in the CND. Comparing these results with those obtained through CBR reveal that the CBR does not take into account the values of frequency biases for the involved KB-Units (see Table 3.10). The CBR results are based on similarity-matching and thus contain those KB-Units too where the frequencies are negative. The model developed here allows future enhancements and has applications in a number of fields from AI games to simulation based trainings for offshore emergency situations. Future work should consider other measures of similarity as mentioned in Section 3.3. Finally, integration of the BDI agent model may be an interesting area of investigation.

References

Alterman, R. (1988). Adaptive planning. *Cognitive Science*, *12*, 393–421.

Anderson, M. & McCartney, R. (2003). Diagram processing: Computing with diagrams. *Artificial Intelligence*, *145*(1-2), 181–226. doi:http://dx.doi.org/10.1016/S0004-3702(02)00383-1

BP.com. (2010, September 8). Deepwater horizon accident investigation report [Web Posting]. Retrieved from http://www.bp.com/liveassets/bp_internet/globalbp/globalbp_uk_english/incident_response/STAGING/local_assets/downloads_pdfs/Deepwater_Horizon_Accident_Investigation_Report.pdf

Brooks, R., Arbel, T., & Precup, D. (2008). Anytime similarity measures for faster alignment. *Computer Vision and Image Understanding*, *110*(3), 378–389. doi:http://dx.doi.org/10.1016/j.cviu.2007.09.011

Buckland, M. (2005). *Programming game AI by example*. Sudbury, MA: Wordware Publishing Inc.

Chen, T.-H., Wu, P.-H., & Chiou, Y.-C. (2004). An early fire-detection method based on image processing. In *ICIP*, October 24–27, 2004 (Vol. 3, pp. 1707–1710). Singapore.

Christou, M. & Myrto, K. A. (2012). *Safety of offshore oil and gas operations: Lessons from past accident analysis: Ensuring EU hydrocarbon supply through better control of major hazards*. Luxembourg: JRC Scientific and Policy Reports, Publications office of the European Union.

Duch, W. (2000). Similarity-based methods: a general framework for classification, approximation and association. *Control and Cybernetics*, *29*(4).

Fagin, R., Kumar, R., & Sivakumar, D. (2003). Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 acm sigmod inter-*

national conference on management of data (pp. 301–312). SIGMOD '03. San Diego, California: ACM. doi:10.1145/872757.872795

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison Wesley Longman Publishing Co., Inc.

James, W. (1908). *Talks to teachers on psychology: And to students on some of life's ideals*. London, UK: Longmans, Green & Co.

Khakzad, N., Khan, F., & Amyotte, P. (2013). Quantitative risk analysis of offshore drilling operations: A bayesian approach. *Safety Science*, *57*, 108–117. doi:10.1016/j.ssci.2013.01.022

Khan, F., Sadiq, R., & Hussain, T. (2002). Risk-based process safety assessment and control measures design for offshore process facilities. *Journal of Hazardous Materials*, *94*(1), 1–36. doi:http://dx.doi.org/10.1016/S0304-3894(02)00004-3

Klein, G. (1998). *Sources of power: how people make decisions*. Cambridge, MA: MIT Press.

Knuth, D. E. (1969). *The art of computer programming: seminumerical algorithms*. Reading, MA: Addison Wesley Longman Publishing Co., Inc.

Kruetzer, W. & McKenzie, B. (1991). *Programming for artificial intelligence: Methods, tools and application*. Reading, MA: Addison-Wesley.

Lincoln, N. K. & Veres, S. M. (2013). Natural language programming of complex robotic BDI agents. *Journal of Intelligent & Robotic Systems*, *71*(2), 211–230.

Majumdar, P. & Samanta, S. K. (2013). Decision making based on similarity measure of vague soft sets. *Journal of Intelligent & Fuzzy Systems*, *24*, 637–646.

Matsumoto, M. & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions*

on Modeling and Computer Simulation - Special issue on uniform random number generation, *8*(1), 3–30. doi:10.1145/272991.272995

Millington, I. & Funge, J. (2009). *Artificial intelligence for games* (2nd ed.). Burlington, MA: Morgan Kaufmann Publishers.

Musharraf, M., Hussain, J., Khan, F., Veitch, B., MacKinnon, S., & Imtiaz, S. (2013). Human reliability assessment during offshore emergency conditions. *Safety Science*, *59*, 19–27. doi:10.1016/j.ssci.2013.04.001

Musharraf, M., Khan, F., Veitch, B., MacKinnon, S. N., & Imtiaz, S. (2013). Human factor risk assessment during emergency condition in harsh environment. In *Proc. of the 32nd international conference on ocean, offshore and Arctic engineering (OMAE), June 9-14, 2013, Nantes, France*, June 9–14, 2013. Nantes, France.

NFPA.org. (2013, August 12). NFPA 10: Standard for portable fire extinguishers [Online]. Retrieved from http://www.nfpa.org

Rao, A. S. & Georgeff, M. P. (1995). BDI agents: from theory to practice. In L. Gasser & V. Lesser (Eds.), *Proceedings of the first international conference on multi-agent systems*: (pp. 312–319). San Francisco, CA.

Reason, J. (1990). *Human error*. Cambridge, UK: Cambridge University Press.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In *Proceedings of game developers conference 1999 san jose* (pp. 763–782). San Francisco, CA: Miller Freeman Game Group.

Russel, S. J. & Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *2*, 575–609. doi:10.1613/jair.133

Santini, S. & Jain, R. (1996). Similarity matching. In S. Z. Li, D. P. Mital, E. K. Teoh, & H. Wang (Eds.), *Proceedings of second Asian conference on computer vision, ACCV '95 Singapore, December 5-8, 1995. Invited session papers, re-*

cent development in computer vision (Vol. 1035, pp. 570–580). Lecture Notes in Computer Science. Heidelberg: Springer-Verlag.

Skogdalen, J. E., Khorsandi, J., & Vinnem, J. E. (2012). Evacuation, escape, and rescue experiences from offshore accidents including the deepwater horizon. *Journal of Loss Prevention in the Process Industries*, *25*(1), 148–158. doi:10.1016/j.jlp.2011.08.005

Sloman, A. (1995). Musings on the roles of logical and non-logical representations in intelligence. In J. Glasgow, N. Narayanan, & B. Chandrasekaran (Eds.), *Diagrammatic reasoning*: *Cognitive and computational perspectives*. Cambridge, MA: MIT Press.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., & Jain, R. (2000). Content based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(12), 1349–1380. doi:http://dx.doi.org/10.1109/34.895972

Stentiford, F. (2007). Attention-base similarity. *Pattern Recognition*, *40*(3), 771–783. doi:http://dx.doi.org/10.1016/j.patcog.2006.05.014

Sullivan, L. E. (2009, December). *The SAGE glossary of the social and behavioral sciences*. Thousand Oaks, California: SAGE Publications Inc. doi:10.4135/9781412972024

Swain, A. D. & Guttmann, H. E. (1983). *Handbook of human reliability analysis with emphasis on nuclear power: final report*. Washington: US Nuclear Regulatory Commission. Retrieved from http://books.google.ca/books?id=BCG7QwAACAAJ

Tompkins, O. (2010). Panic attacks. *AAOHN Journal*, *58*(6), 268. doi:10.3928/08910162-20100526-07

Tversky, A. (1977). Features of similarity. *Psychological Review*, *84*(4), 327–352. doi:http://dx.doi.org/10.1037/0033-295X.84.4.327

Verstockt, S., Hoecke, S. V., Tilley, N., Merci, B., Sette, B., Lambert, P., ... Walle, R. V. (2011). Firecube: a multi-view localization framework for 3d fire analysis. *Fire Safety Journal*, *46*(5), 262–275. doi:10.1016/j.firesaf.2011.03.001

Verstockt, S., Vanoosthuyse, A., Hoecke, S. V., Lambert, P., & Walle, R. V. (2010). Multi-sensor fire detection by fusing visual and non-visual flame features. In A. Elmoataz, O. Lezoray, F. Nouboud, D. Mammass, & J. Meunier (Eds.), *Proceedings of the 4th international conference, ICISP 2010, Trois-rivières, QC, Canada, June 30-July 2, 2010, image and signal processing* (Vol. 6134, pp. 333–341). Lecture Notes in Computer Science. Heidelberg: Springer-Verlag.

WAtoday.com.au. (2009, August 29). Oil spill off WA coast much larger than reported: greens [Online Newspaper]. Retrieved from http://www.watoday.com.au/wa-news/oil-spill-off-wa-coast-much-larger-than-reported-greens-20090828-f2i7.html

Wooldridge, M. J. & Jennings, N. R. (1995). Agent theories, architectures and languages: a survey. In M. J. Wooldridge & N. R. Jennings (Eds.), *Lecture notes in computer science*: *Vol. 890. Intelligent agents. proceedings ofECAI-94 workshop on agent theories, architectures, and languages amsterdam, the netherlands, august 8-9, 1994* (pp. 1–39). Heidelberg, Germany: Springer-Verlag. doi:10.1007/3-540-58855-8

Yan, W., Zanni-Merk, C., Rousselot, F., & Cavallucci, D. (2013). Ontology matching for facilitating inventive design based on semantic similarity and case-based reasoning. *International Journal of Knowledge-based and Intelligent Engineering Systems*, *17*, 243–256.

Chapter 4

On slips, surprise and ignorance

At the onset of a decision making state

Authors: Syed Nasir Danial, Faisal Khan, Brian Veitch, and Scott MacKinnon.

Submitted to: Cognitive Systems Research.

Date of Submission: October 1, 2013.

The effects of mental stress and distraction, in terms of making slips, caused by simulated harsh weather conditions are presented for an intelligent agent model suitable for a virtual environment to train people for emerging hazards. Alarms are introduced due to the importance of their timely detection and identification in emergency events like fire hazards. Here, a system is proposed by virtue of which the agent can make slips. To further our insight into the agent's pre- and post-decision mental state, *surprise* and *ignorance* are estimated. The agent is surprised if it makes a less likely decision out of a number of more likely ones. The state of being surprised is the post-decision state. The ignorance estimates how informed the agent is towards making a particular decision, and that is the pre-decision state.

Keywords: agents making slips, cognitive error-types, slips, surprise, ignorance

4.1 Introduction

A number of agent models and frameworks have been in use for different domains, but one that could work in a VE that models harsh environmental and emergency situations, such as those on offshore oil & gas platforms, is discussed here.

Reason (1990) reports that slips, which occur at the cognitive stage of execution, are responsible for a number of incidents on various sensitive installations. For instance, at the Three Mile Island Nuclear disaster (Kemeny, 1979; Reason, 1990) the block valves were erroneously left in a wrong position and the valve status indicator was covered by a nearby maintenance tag. Design of control rooms and instruments therein have significant importance (Norman, 2002) as far as human perception through any instrument is concerned because increasing stress, distraction, and time constraint in emergencies practically leaves no room for rational thinking. Similarly, at the King's Cross Underground fire case, London (1987), the Relief Station Inspector fetched the fire extinguisher but could not use it as he was thinking to activate the water fog system — his mind was pre-occupied by other thoughts (Reason, 1990). An operator in a control room may imagine that the control valve is closed but in reality it may not be the case, he can forget due to high stress or distraction. Detection of alarms or their identification may also be misunderstood. Musharraf et al. (2013) report that detection and identification of alarms in a mustering event on an offshore facility are dependent on stress and distraction. They also identify other PSFs which are not considered here.

This article proposes an algorithm for the intelligent agent model reported elsewhere (Danial, Khan, Veitch, & MacKinnon, 2013). The algorithm is used to construct the knowledge-retrieving CND for the agent to the extent that the agent can model

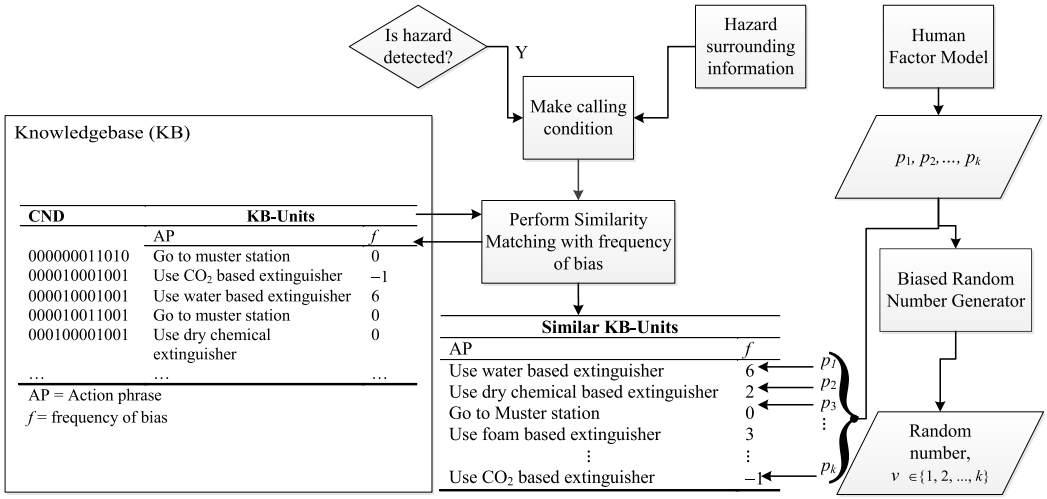

Figure 4.1: The agent model. The values in the KB and the Similar KB-Units are only for illustration purposes.

slips especially when its stress and distraction are high. Two types of alarms are introduced, viz., the process alarm (PAL) and the muster alarm (MAL). We introduce here two measures to estimate the agent's *surprise* and *ignorance* in a typical decision-making situation. The agent is tested in several simulations and it is shown that the proposed algorithm introduces slips effectively under high stress and distraction. The results of the simulations show that the agent, with the proposed algorithm for constructing a retrieving CND, has potential to model effective human behaviour in VEs for emergency scenarios, such as emerging fire hazards.

4.2 The agent model and scenario analysis

A brief description of the agent model, presented in Danial et al. (2013) and shown in Figure 4.1, is being given in this section. The agent's KB is an associative database (Kruetzer & McKenzie, 1991) of KB-Units associated with CNDs. The CND is a

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F	Ex	B	HL_1	HL_2	HL_3	U	O	El	Fu	M	S	FC_1	FC_2	FC_3	A_1	A_2
1	0	0	1	0	0	0	0	1	1	0	0	1	1	0	1	0

Figure 4.2: The calling condition model. The bit labeled 'F' is set if fire hazard is detected. HL_1, HL_2 & HL_3 are for hazard level. Bits 'O', 'El', 'Fu', 'M' & 'S' represents ordinary combustibles, power lines, fuel lines, metal dust and smoke respectively. Bits FC_1, FC_2, FC_3 keeps fire class information and bits A_1 & A_2 are used for process and muster alarms respectively.

bit-field (see Figure 4.2) with certain fixed bits representing the presence or absence of an entity or event. It is used as a *key* to retrieve required KB-Units, those that contain information about what to do in the current situation. The KB-Unit is a data structure having two member variables — an AP and `f`. The `f` keeps the count of the use of a particular AP in the past under the associated CND. However, the $\texttt{f} = -1$ means that the AP is *not suitable* under the associated CND. The AP is a phrase that refers to an action to be performed by the agent. The APs used here are shown in Table 4.1. The Human Factor model (see Figure 4.1) estimates the agent's reliability, p, on the basis of its stress and distraction. The distraction is controlled by varying the input parameters *rain* and *wind* which are scripted as the weather factors whereas the parameter stress is provided directly without involving other factors.

By default the agent wanders around the VE, which is a 2D graphical environment, containing walls, a muster station, combustible metals, obstacles like chairs, electrical lines, fuel lines, and fire extinguishers installed at different fixed locations. Fire objects update their temperature, flame length and flame width on each update-cycle of the environment. A navigation graph is used to construct paths from one position to another. A fire hazard is classified as either low or high. For a low hazard, the selection of an appropriate fire extinguisher is essential in fire fighting as described by the NFPA-10 standards (NFPA.org, 2013). NFPA classifies fire as: A, B, C, D, and

K. The class A fire is due to burning of ordinary combustibles such as wood, paper, cloth, rubber or plastic. The class B fire is caused by igniting flammable liquids. The class C fire is due to energized electrical equipments. The class D fire is due to certain metals and the class K fire is by burning cooking medium.

After observing a fire hazard, the agent recognizes the surrounding objects and generates a retrieving CND. The retrieval process takes this CND as input and produces a set of distinct KB-Units at the output (shown as 'Similar KB-Units' in Figure 4.1) by exploiting the concepts of similarity-matching and frequency bias (Reason, 1990). This set of 'Similar KB-Units' is sorted according to the Hamming distance between the retrieved and stored CNDs, and values of `f`. Also, the redundant KB-Units are removed and only the most similar and frequent KB-Units are kept. Now if there are k KB-Units in the set of 'Similar KB-Units', the Human Factor model returns k probability (reliability) values, $p_1 = p, p_2, \ldots, p_k$ (see Section 4.3). These reliability values are assigned to each of the KB-Units in the set. Besides that, $p_1, p_2, \ldots, p_k$ are also used to draw a random number, ϑ, from a biased random number generator (Matsumoto & Nishimura, 1998) such that $1 \leq \vartheta \leq k$. The agent selects the KB-Unit that has the index ϑ in the KB, fetches the value of AP and then selects the goal that is associated with this AP (see Table 4.1). When the goal is selected the agent performs the actions programmed in each goal class.

As an example, consider the case when the agent, at an initial position $P_1(a_1, b_1)$, observes a high level fire hazard and immediately decides to go to the muster station by selecting the KB-Unit wherein AP contains `GM`. The agent selects the goal `GOAL_GM` which is associated with `GM`. Now the goal object will determine the location of the muster station by allocating a seat. The position, say $P_2(a_2, b_2)$, of the seat is used as the target in the A* search (Buckland, 2005) that returns a least cost

Table 4.1: List of APs with associated goal classes

AP	Description	Goal classes
UW	Use water-base extinguisher	GOAL_UW
UF	Use foam-base extinguisher	GOAL_UF
UC	Use CO_2-base extinguisher	GOAL_UC
UD	Use dry chemical base extinguisher	GOAL_UD
UP	Use purple-K dry chemical extinguisher	GOAL_UP
UM	Use multipurpose dry chemical extinguisher	GOAL_UM
UDP	Use dry powder extinguisher	GOAL_UDP
UWC	Use wet chemical extinguisher	GOAL_UWC
GM	Go to the muster station	GOAL_GM
CP	Continue previous goal	GOAL_CP
PA	Pay attention	GOAL_PA

Algorithm 5: Generate k reliability values.

$pr[1] = p$;
for I=*1 to* $k-1$ **do**
    $pr[I+1]$=$p \times (1-p)^I$;
end
$pr[k]$=$(1-p)^{k-1}$;

path, V, from P_1 to P_2. The path V is passed to the GOAL_FOLLOW_PATH class where each subsequent update cycle will bring the agent closer and closer to its allocated seat in the muster station. In this way, the goal object GOAL_GM uses another goal object, GOAL_FOLLOW_PATH that implements the steering behavior *path follow* (Reynolds, 1999; Buckland, 2005), and achieved what was intended in the AP by its value GM or "Go to the muster station".

4.3 Estimating the Agent's Reliability

The present work uses two stressors, (*i*) *mental stress*, and (*ii*) *distraction*, rather than *panic*, *fear* and *over confidence bias* as reported in Danial et al. (2013) because offshore oil and gas installations are largely affected by harsh weather conditions, heavy rains

Table 4.2: Estimates of the agent's reliability p for various ranges of the values of r

r	p	r	p
≥ 8	1.00	[4, 5)	0.60
[7, 8)	0.90	[3, 4)	0.55
[6, 7)	0.80	≥ 2	0.50
[5, 6)	0.70	< 2	0.40

and strong winds, that induce stress and distraction among people (Musharraf et al., 2013). How reliable is the agent's decision in a given situation is estimated through the agent's reliability index, r, such that

$$r = -(\log stress + \log distraction), \quad 0 < stress \leq 1, \quad 0 < distraction \leq 1. \quad (4.1)$$

The estimate of agent's reliability, p, which is the probability that the agent acts accordingly, is given in Table 4.2. Following Algorithm 5, k reliability values are generated and assigned to the retrieved KB-Units and a random index is generated which is used to select a KB-Unit from the retrieved set of KB-Units for later deliberation as explained in the previous section.

4.4 *Slip* enabled retrieving CND algorithm

The algorithm for constructing a CND that advances the agent model so much so the agent can now make slips in stressed behavior is described here in the form of flowcharts in Figure 4.3 and 4.4. It is implemented as a method `MakeCallingCondition` mentioned in Figure 4.1. When the agent observes a fire hazard, it initializes a new CND object, which is an object of a C++ class in current implementation, with zero and determines the neighbourhood size, ϵ, of the fire object. Now if the agent is reliable, which means if the stress and distraction are either zero or negligible, then

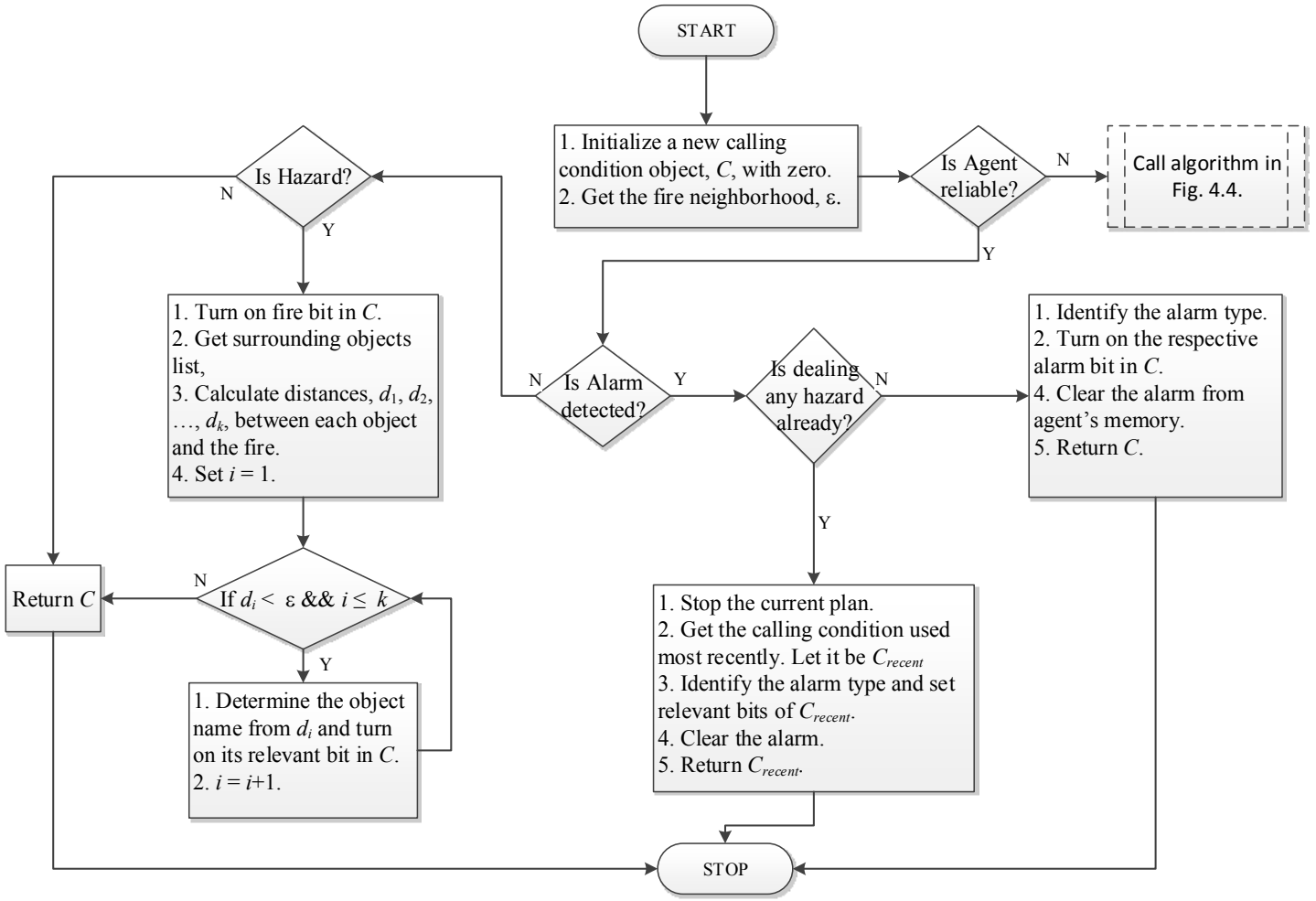

Figure 4.3: Part-I of the flowchart to construct the retrieving calling condition.

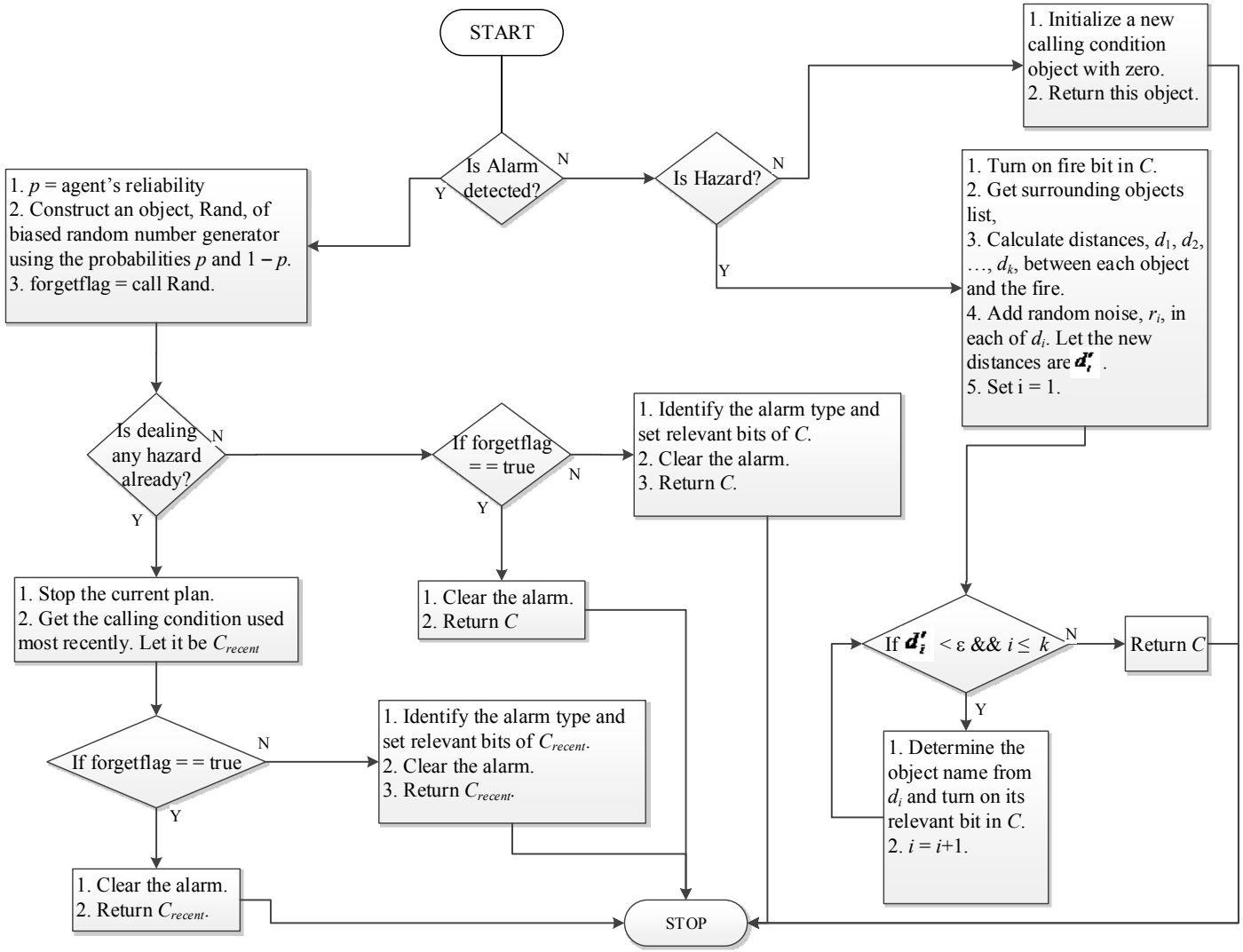

Figure 4.4: Part-II of the flowchart to construct the retrieving calling condition.

the algorithm does not perform anything that could introduce slips. So, the CND would have information about hazard, and its surrounding objects such that the distance, d_i, of the ith surrounding object from the hazard object is less than ϵ. Similarly, depending upon whether the agent received an alarm signal before or after the detection of the hazard, the algorithm enters its information in the CND and returns the CND to the following module for retrieval of KB-Units as depicted in Figure 4.1.

When the agent is stressed its reliability would be low. If it detects a fire hazard without any alarms, and if there are M surrounding objects then it calculates the distances $d_i, \forall i \leq M$ and adds a random noise ξ_i in each of the respective distances d_i. In this way the new distances, d'_i, may not be in the same relation as $d_i \leq \epsilon, \forall i \leq M$ if the old distances are replaced by the new ones. Thus, some of the objects — lying within ϵ distance from the fire — would not be considered, and the others which lie slightly outside the ϵ may be considered inside. Now, the algorithm sets the corresponding bits in the CND (see Figure 4.2) for the objects for which $d'_i \leq \epsilon$ holds and returns the CND to the later phases of the agent model.

Similarly, when the agent detects an alarm, the algorithm first gets a biased random Boolean value, ϑ_b, by employing the Mercenne Twister algorithm (Matsumoto & Nishimura, 1998) using p and $1 - p$ as the biasness criteria, and then determines if there is already a hazard detected or only the alarm is detected. In the former case, the algorithm fetches the CND which is being used for the current hazard and updates its value by either including the alarm information in the CND or leaving the CND unchanged. The decision whether to include the alarm information or not depends on the value of ϑ_b . If $\vartheta_b =$ `true` the CND would render unchanged, and if $\vartheta_b =$ `false` then the alarm related bits in the CND will be set. In the later case,

i.e., when the alarm was received but the agent was not involved in any hazardous situation, then depending on the value of ϑ_b the alarm bits will be set or cleared in the zero initialized CND object. In both of these cases, the CND object will either have the alarm information or not depending on the value of ϑ_b and the updated CND is sent to the knowledge retrieval phase for retrieving the KB-Unit. Since ϑ_b is obtained from the biased random number generator, the agent's reliability has a vital role in the final decision about whether the agent will neglect the alarm or not.

4.5 Agent's surprise and ignorance

The term 'surprise' was first coined by Myron Tribus in his 1961 book *Thermostatics and Thermodynamics* in the context of information theory. Tribus (1961) says that the amount of self-information, i.e. $\omega(p_i)$ in Equation 4.7, can also be used to estimate surprise. This value causes surprise to an observer. In the present study, the agent also acts as an observer of the reactions of the environment against its own actions. A surprising action is definitely the one that is less probable due to its inappropriateness for some given conditions. This inappropriateness is highly likely to cause unexpected results which in turn brings surprise to the agent in response to observing the reactions after its own actions. However, there is no need to delay the estimation of the surprise (till the time when the agent observes the reaction) because it can be calculated right at the time when the action is choosen.

In order to formalize the notion of surprise let us consider an AP as a proposition because either it is selected (`true`) or not selected (`false`). Also, when one is selected, it becomes `true` as the agent performs it thereby rendering every other `false`. Thus, if a proposition symbol θ represents an AP the output of the KB-Units retrieval

algorithm, i.e., the Algorithm 1, is given by:

$$\Psi = \{\theta_1, \theta_2, \ldots, \theta_n\}, \tag{4.2}$$

such that:

— No two θ's can be `true` (or selected) simultaneously, that implies:

$$\theta_i \wedge \theta_j = \texttt{false}, \quad i \neq j. \tag{4.3}$$

— At least one of the propositions must be true (or selected), implying:

$$\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n = \texttt{true}. \tag{4.4}$$

Therefore, Ψ is a disjoint and exhaustive set, sometimes called a logical spectrum (Park & Band, 1976; Watanabe, 1969). Now the probability $\Pr\{\theta_i\}$ of the proposition θ_i can be considered as the degree of expectation, before selection, that θ_i will be the case taken up by the agent. The function $p(\theta)$ is called probability mass function if a real number $p(\theta_i)$ can be attached to each of the n constituent members of Ψ in such a way that

$$p(\theta_i) \geq 0, \tag{4.5}$$

and

$$\sum_{i=1}^{n} p(\theta_i) = 1. \tag{4.6}$$

Algorithm 2 fulfills the conditions in Equation 4.5 and 4.6 in assigning probabilities to the propositions (or APs) θ in the set Ψ obtained by the retrieval process.

The θ_is are experimentally testable propositions, retrieved and sorted according to

similarity matching and frequency bias against a retrieving CND. This means, in a given situation, θ_i is a more legitimate solution than θ_{i+t} for some $t > 0$ and p_i determines its chances of becoming true.

The selection is made by generating a random number, ϑ, such that $1 \leq \vartheta \leq n$, through a biased random number generator that uses p_is to incorporate the biasness (Danial et al., 2013). The agent's surprise is an estimate of unlikely selection of those θ_i that had very weak chances of becoming true. We represent surprise as a function ω (Watanabe, 1969) such that,

$$\omega(p_i) = -\log p_i, \quad 0 < p_i \leq 1. \tag{4.7}$$

where a logarithm to any base can be used. Thus if a certain proposition (or AP) such as, `UW`, had the probability 0.05 and the agent selected it (that makes it true) then the associated surprise would be $\omega(0.05) = -\log_e(0.05) \approx 3.0$, which is a big surprise. The lesser the value of the probability p_i of an AP before selection (that turned out to be true) the higher the associated surprise. For the purpose of ease in interpreting the values of ω, the following transformation is used to scale the results down to the interval $[0, 1]$:

$$\Gamma(p_i) = \frac{(\omega(p_i) - \omega_{min})}{(\omega_{max} - \omega_{min})} = \frac{(\omega(p_i) - \omega(p_{max}))}{(\omega(p_{min}) - \omega(p_{max}))}, \quad \omega_{max} \neq \omega_{min} \tag{4.8}$$

where, $p_{min} = \min_i p_i$ and $p_{max} = \max_i p_i$, are the minimum and the maximum probabilities assigned to any two propositions in a single experiment. In this way, $\Gamma(p_i) = 0$ means 'no surprise' and a $\Gamma(p_i) = 1$ means maximum surprise. The values of $\Gamma(p_i)$ in the open interval $(0, 1)$ will, therefore, have intermediate levels of surprise.

Also, $\omega(p_i)$ is a random variable with probability p_i. Thus its expected or average

value before selecting θ_i is given by:

$$E(\omega(p_i)) = E(\omega) = -\sum_{i=1}^{n} p(\theta_i) \log p_i. \tag{4.9}$$

The Equation 4.9 measures the agent's ignorance (Watanabe, 1969) in certain state of knowledge before making any decision. Thus,

$$ign(\Psi) = E(\omega) = -\sum_{i=1}^{n} p(\theta_i) \log p_i. \tag{4.10}$$

The minimum of $ign(\Psi)$ occurs when one of the p_is becomes unity and the rest are zero. Theoretically, $\min ign(\Psi) = 0$ and that corresponds to the case when the agent is sure about one of the θ_i before actually selecting it. Similarly the maximum of $ign(\Psi)$ happens when all the p_is are equal to $1/n$. In this case, as all θ_i have equal chance of being selected, the agent would be completely ignorant about what would happen. Thus, the function $ign(\Psi)$ is bounded as:

$$0 \leq ign(\Psi) \leq -\log \frac{1}{n}. \tag{4.11}$$

Note that the bounds do not involve p_is. Equation 4.8 can also be used to scale the values of ign in Equation 4.10 to the closed interval $[0, 1]$. Clearly, the function $ign(\Psi)$ tells us about the extent to which our expectation of the occurrence of propositions is widely scattered over various propositions. In other words, ignorance is the expected surprise and that does mean how ignorant the agent would be from the likelihood of choosing the best available option. This, however, does not suggest the inadequacy of the KB even for the cases when the agent's mental state is not normal because in such cases even if the retrieval algorithm brings the best KB-Unit at the top of the list, the low reliability value of the agent would lower the chances of selecting the top

level KB-Units.

4.6 Scenarios and results

The agent model is implemented as a C++ program. Here before initiating the KB retrieval process the agent uses flowcharts of Figure 4.3 and 4.4 to construct the retrieving CND. The surprise and ignorance metrics are computed for each of the scenarios being presented here.

4.6.1 Scenario 1: Stressed agent facing fire, rain and wind

The agent was under high stress and was situated in the VE. The agent faced rain and wind both and then a sudden fire hazard due to burning ordinary combustibles. The following two simulations were performed:

4.6.1.1 First simulation

The agent was situated in the VE and observed a low level fire hazard but miscalculated the nearby ordinary combustibles. The fire class information was sent to the agent explicitly, thus the agent, under high stress (or low reliability, $p = 0.4$), quickly found out a list of KB-Units: `UW`, `UF`, `UM`, `GM`, `UDP`, `UC`, `UD`, `UP`, `UWC`. The value of `f` for the first five of these KB-Units was 0 and for the last four KB-Units was -1. The respective probabilities assigned to each of the KB-Units above were 0.40, 0.24, 0.14, 0.09, 0.05, 0.03, 0.02, 0.01, and 0.02. The ignorance $ign(\Psi) = 1.65$ (or $\Gamma(ign(\Psi)) = 0.75$ means the agent was 75% ignorant. In other words, the agent was not sure about (*i*) a unique best solution, (*ii*) a unique worst solution, before making

a choice. However, the agent was more inclined in making a choice from the first two or three KB-Units. It selected the second KB-Unit `UF` and had $\Gamma(\omega(p_2)) = 0.143$ that means the agent was 14.3% surprised or somewhat surprised! The fire was extinguished due to the application of foam based extinguisher and a new KB-Unit was added successfully in the KB.

4.6.1.2 Second simulation

Here also, the agent could not identify the presence of nearby ordinary combustibles. Thus the retrieving CND was the same as in the case reported in the previous simulation. This time an exact match from the KB was found because the agent had updated the KB previously. However, due to low reliability, the search space included similar KB-Units too. Overall, the KB search resulted in: `UF`, `UM`, `UW`, `GM`, `UDP`, `UWC`, `UP`, `UD`, and `UC`, where only `UF` had $\mathtt{f} = 1$. The probability assignment was in similar fashion as before, but the agent selected `UM` and was then 14.3% surprised that it selected the one with lesser chances of being correct. The KB was updated appropriately.

4.6.2 Scenario 2: Stressed agent facing fire, rain and wind in the presense of alarms

The agent was under high stress facing rain and wind and then a sudden fire hazard was observed. The fire was due to burning of ordinary combustibles. Then the agent recieved alarms. The following different simulatioin were performed:

4.6.2.1 First simulation

The PAL was sent three times before the agent could observe any hazards but the agent was stressed so it ignored the alarms and continued wandering in the VE until it encountered the fire hazard. The KB search retrieved the KB-Units: `UM`, `UF`, `UW`, `GM`, `UDP`, `UC`, `UD`, `UP`, and `UWC` with respective probabilities 0.4, 0.24, 0.144, 0.086, 0.052, 0.031, 0.019, 0.011, and 0.017. The agent selected the `UF` and extinguished the fire with a surprise of 14.3%.

4.6.2.2 Second simulation

First, the agent received the PAL, remembered it, and found two relevant KB-Units: `PA` with $p = 0.4$ and `GM` with $p = 0.6$. The ignorance was around 97% suggesting almost no difference in making a specific choice. The agent selected `GM`, which was the wrong choice. The agent chose this wrong choice based on its probability, so it could not be surprised —$\Gamma(\omega(0.6)) = 0$ — as it has chosen the most expected KB-Unit.

4.6.2.3 Third simulation

The PAL was sent to the agent and that led it chose `PA`. The plan of `PA` held the agent waiting for a muster alarm. When the muster alarm came, the agent searched the `KB` and found `PA` and `GM`. It selected `GM` with no surprise — $\Gamma(\omega(0.6)) = 0$ — and reached the muster station.

4.6.2.4 Fourth simulation

The PAL was sent to the agent before observing any hazards and it chose `GM`. The agent was not surprised because $\Gamma(\omega(0.6)) = 0$ as the corresponding $p = 0.6$, which means that at that particular moment, `GM` was comparatively a better solution according to the agent.

4.6.2.5 Fifth simulation

The agent detected a fire hazard, also marked ordinary combustibles, searched the KB and found: `UW`, `UF`, `UM`, `CP`, `GM`, `UC`, `UD`, `UP`, `UDP`, `UWC`. The first three KB-Units were obtained through exact match. `UW` was selected and surprise was zero. Before making the decision, the agent was a bit more inclined towards the first solution compared to the other KB-Units, but the ignorance metric, $\Gamma(ign(\Psi)) = 72.3\%$, says that the difference among most of the KB-Units, as far as their selection chance is concerned, was not marginal. Here, the probabilities assigned to each of the KB-Units given above are: 0.4, 0.24, 0.14, 0.09, 0.05, 0.03, 0.02, 0.01, 0.01, and 0.01 respectively. Now at this time, when the agent started executing the plan of `UW`, it received a process alarm, recognized it and then combined this information in order to make a new retrieving CND and then used the new CND to find out what to do. `CP` is an `AP` in the KB that means to continue what was in progress. In situations in which a process alarm is identified after low-level hazard detection the agent simply sticks to its previous decision and simply does nothing special in light of the process alarm. But this logic is set in the case when the agent's mental state is normal. In the current case, the agent's high stress changed the plot. The KB produced a list of KB-Units: `CP`, `UW`, `UF`, `UM`, `GM`, `PA`, `UC`, `UD`, `UP`, `UDP`, `UWC`, where `CP` is the only KB-Unit with exact match. The last five had `f` $= -1$. The agent selected `UF` with a bit surprise (24.4%).

Table 4.3: Results for Section 4.6.2.6 on different runs of the simulation. The alarm value 'Yes' means the alarm was identified and the value 'No' means that the alarm was ignored.

Before PAL			Alarm	After PAL			After MAL			Alarm
AP	$\Gamma(\omega)$	$\Gamma(ign)$		AP	$\Gamma(\omega)$	$\Gamma(ign)$	AP	$\Gamma(\omega)$	$\Gamma(ign)$	
UF	12%	49%	Yes	CP	0%	49%	CP	0	49%	No
UW	0%	49%	Yes	GM	69%	86%	-	0	-	-
UW	0%	49%	No	UW	0%	49%	UW	0	49%	No
UF	35%	49%	Yes	CP	0%	86%	GM	0	86%	Yes
CP→UW	23%	49%	No	UW	0%	49%	UW	0	49%	No

The experiment was repeated five times. The second time, the agent miscalculated the nearness of ordinary combustibles, selected UW but could not focus on the process alarm so did nothing against the alarm. However, it extinguished the fire safely. The third time, the agent again made a wrong judgment about the fire surroundings, selected UM with a surprise of 12.2%, then it received the process alarm, searched the KB again and found CP, UW, UM, UF, GM, UWC, UDP, UP, UD, and UC in order. The agent selected CP with no surprise, $\Gamma(\omega(0.4)) = 0$. The selection of CP allowed the agent to continue the previous goal so it remained with UW and used the water based extinguisher to extinguish the fire successfully. The fourth time the agent selected GM from UW, UF, UM, CP, GM, UDP, UC, UD, UP, UWC with $p_5 = 0.05$ and thus a big surprise of 55.5%. Certainly this was a wrong choice and the agent made it because of its low reliability. The last time, the agent selected UF (which was the third best choice), ignored the alarm and regenerated the calling condition that resulted in the selection of UM.

4.6.2.6 Sixth simulation

Here the stress is reduced and thus the agent reliability, p, is changed to 0.6. The process alarm is sent after observing the low level fire hazard and the muster alarm

Table 4.4: Results of Section 4.6.3 on different runs of the simulation.

Alarm remembered?	Cases	KB-Units found	KB-Unit selected	$\Gamma(\omega)$	$\Gamma(ign)$
Yes	64	PA,GM	PA	0	72.5%
No	12	-	-	-	-
Yes	11	PA,GM	GM	1	72.5%

is sent when the agent picked up a fire extinguisher and is on its way to the hazard location. We performed this experiment five times and the results are shown in Table 4.3.

4.6.3 Scenario 3: A bit stressed agent, recieved alarms

The agent was under low stress and recieved alarms before detecting any hazards. The agent's reliability was 0.8, when it received the PAL. The experiment was repeated 87 times and in 73% cases the agent performed as required, i.e., it selected the KB-Unit PA. Table 4.4 shows the results for each experiment.

In some cases, where the agent had selected PA, we tested the agent by sending the MAL, and in most of the attempts the agent selected GM and went to the muster station. A single instance was that in which more than one muster alarms were sent and then the agent came out of the PA's plan and selected GM.

4.6.4 Scenario 4: Agent with no stress & distraction facing fire, recieved the alarms

The agent was under normal mental state and was situated in the VE when a sudden fire hazard due to burning ordinary combustibles occurred. The agent also responded over alarms. The following simulations were carried out.

4.6.4.1 First simulation

The agent detects the PAL before observing the fire hazard. The KB search delivered `PA` as an exact match. The agent reliability was 1.0 and therefore when the MAL is sent the agent also detected it, abandoned the `PA`'s plan, and went to the muster station. The agent's ignorance value was 0 and the maximum ignorance was also zero. The surprise in both decisions was also zero.

4.6.4.2 Second simulation

In this simulation the PAL was sent after making a decision `UW` against a low level fire hazard detection. The agent combined this alarm information with the currently used retrieving CND (that is, the CND against the KB-Unit `UW` before detecting the alarm) and found an exactly matched KB-Unit: `CP` (whose plan instructs the agent to continue what it is doing). The values of surprise and ignorance were zero for both `UW` and `CP`. When the goal against `CP` was launched, it resumed the plan of `UW`. In simple words, the PAL was detected but did not change the agent's previous decision, though there was a little pause during detection of the alarm and resumption of `UW`'s plan.

4.7 Conclusion

The results show that the agent with high reliability makes decisions in accordance with similarity-matching and frequency bias. The reliability value affects the chances the agent forgets an alarm or estimates the wrong distance between an obstacle and the hazard's location. Forgetting the alarm renders the agent unable to extract the KB-

Units that specifically deal with the alarms and thus the resulting decision, though taken without considering the alarms but actually in the presence of alarms, are likely wrong, could lead to a disastrous situation. For example, one common unsafe behavior of the stressed agent in most of the MAL events is that the selected decision is other than GM. This is not what the expected response should be. Further, timely identification of an alarm saves time to deal with a controllable hazard which otherwise could turn into an emergency scenario. Consider the cases reported in Table 4.3. Because of similarity matching and the frequency of past use, most of the cases report correct decisions. However, some solutions show relatively large values of surprise. For instance, the fourth case, where the selection of UF was legitimate but since the agent also found more better solutions, it was surprised of its decision. In the fifth case, CP→UW is really surprising because the agent was free before observing the fire. After observing the fire there was no way to implement CP, which means to continue the previous goal. The goal CP attempts to add a previous goal to the goal queue, if one is found. But if it does not find any previous goal with the agent, it simply pops up the second KB-Unit in the list. Thus, in this way, CP→UW means that the agent selected CP and the CP launched UW and that was also a correct solution in that case. It is also clearly seen from Table 4.3 that due to high stress and distraction ($p = 0.60$), the agent identified PAL only in 3 out of 5 cases, and identified the muster alarm in only one case in which it made correct decision by choosing GM. Although, in all five cases, the agent was able to extinguish the fire but its decisions, particularly after muster alarms were not, in general, found to be what were required.

Nonetheless, an agent with normal or a bit stressed mental states shows stability in its decisions. Table 4.4 lists the scenarios in which the agent was tested against alarms. In 73% cases the agent responded correctly when it received the PAL. In some of these 73% cases the agent was also sent the MAL and it responded correctly by selecting

GM.

The approach developed here also provides the agent, correctly in most cases, an ability to identify its own mistakes (after making them) by computing the surprise. It also gives the agent a sense of how 'informed' it is in making a choice by calculating the ignorance. However, we have not fully exploited these conclusive points in the model of the agent, except to calculate their estimates. This work could be treated as an extension of the authors' previous work reported in (Danial et al., 2013) and most of the results are in line with previous results, with the exception that the alarms were not dealt with and the slips due to agent's reliability was not incorporated in the information gathering phase (i.e., the process of generation of retrieving CND).

References

Buckland, M. (2005). *Programming game AI by example*. Sudbury, MA: Wordware Publishing Inc.

Danial, S. N., Khan, F., Veitch, B., & MacKinnon, S. N. (2013, July 4). *Dynamic learning and adapting: An AI based approach to enhance virtual training* [Research article]. See Chapter 3. Manuscript submitted for publication. Memorial University of Newfoundland, St. John's, NL, Canada.

Kemeny, J. G. (1979). *Report of the president's commission on the accident at the Three Mile Island*. Washington, DC.

Kruetzer, W. & McKenzie, B. (1991). *Programming for artificial intelligence: Methods, tools and application*. Reading, MA: Addison-Wesley.

Matsumoto, M. & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions*

on Modeling and Computer Simulation - Special issue on uniform random number generation, *8*(1), 3–30. doi:10.1145/272991.272995

Musharraf, M., Hussain, J., Khan, F., Veitch, B., MacKinnon, S., & Imtiaz, S. (2013). Human reliability assessment during offshore emergency conditions. *Safety Science*, *59*, 19–27. doi:10.1016/j.ssci.2013.04.001

NFPA.org. (2013, August 12). NFPA 10: Standard for portable fire extinguishers [Online]. Retrieved from http://www.nfpa.org

Norman, D. A. (2002). *The design of everyday things*. NY: Doubleday.

Park, J. L. & Band, W. (1976). Mutually exclusive and exhaustive quantum states. *Foundations of Physics*, *6*(2), 157–172.

Reason, J. (1990). *Human error*. Cambridge, UK: Cambridge University Press.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In *Proceedings of game developers conference 1999 san jose* (pp. 763–782). San Francisco, CA: Miller Freeman Game Group.

Tribus, M. (1961). *Thermostatics and thermodynamics: An introduction to energy, information and states of matter, with engineering applications*. New York: Van Nostrand.

Watanabe, S. (1969). *Knowing and guessing: a formal and quantitative study*. NY: John Wiley & Sons, Inc.

Chapter 5

Conclusion and recommendations

5.1 Conclusion

It is evident from the literature review that there are many approaches to model an intelligent agent. The model presented here resembles the CBR model where a decision relies on similarity between the stored and retrieving CNDs. The present model adds extra dimensions by including frequency of bias and considering cognitive attitudes in the decision-making process. The use of frequency information provides two unique perspectives. First, when the KB-Units that are associated with the same CNDs are retrieved they are sorted according to their respective frequency values. Thus, under the normal mental state, the most frequent KB-Unit is retrieved and its associated goal is brought into execution. Second, the frequency information acts as the agent's experience with a particular KB-Unit; the feedback mechanism as described in Algorithm 4 either increments it or sets it to negative unity for undesired choices. Thus the sorting method puts the more frequent KB-Units on top of the data structure holding the retrieved KB-Units.

Moreover, under a stressed mental state, even if the reliability is too low to be natural, there is still enough room for the agent to retrieve those KB-Units that have zero

or greater frequency of bias because Algorithm 1 places all the negative frequency KB-Units near the bottom of the data structure holding the retrieved KB-Units. This claim is supported by the simulation results where even high mental stress does not show absurd behaviour. The simulation results show that under stress a novice agent failed about 9% of the time, whereas a trained agent has been found to exhibit only around 1% failure. Further, mistakes are not only committed when a person is stressed, panicked or has fear, although these factors do enhance the chance of a mistake. In any working environment, such as offshore facilities, people do have a bit of stress, or at least their mental state is not the same as when they are at home. It is shown here that the agent that is in normal mental state may also go for selecting the second or third best choice instead of always making the best solution by employing the retrieved KB-Unit.

The proposed agent-model is extended to incorporate 'slips' in dealing with two specific matters. The first is in computing the distance between an obstacle and a hazard. This enables the agent, in cases when slips occur, to construct an inappropriate retrieving CND by either ignoring some obstacles that are near the hazard or by considering some obstacles near although they are far from the hazard. In this way, the agent under high stress may estimate wrong retrieving CND that would definitely lead to wrong choices from the KB and the reason is an illusion that the agent had due to considering something to be near or far from the hazard while the reality was completely opposite. The second is the matter of responding to alarms. The agent can simply forget an alarm as a result of making a slip. Also, as the alarms are sent as discrete messages it is possible that the agent recognizes the alarms too late. Clearly, in real emergency situations such acts may be disastrous for one's own life and the environment.

5.2 Recommendations

The agent-model presented here has been implemented in C++ programming language as standalone software requiring no back-end support from any database or logic-programming frameworks, such as Prolog. The simulations have shown diverse and useful behaviours, in the presence of biases or moderators, pertaining to training scenarios. Still, the author believes that:

1. The model should be used in a real training program, such as AVERT, and the response from human participants should be recorded. Such an exercise would produce insight into how to make new scenarios or what other types of mental simulations are required for effective learning.

2. The concept of *recency* (Reason, 1990) should play an important role in future advancement of the model.

3. Integration of the BDI-agent model is an interesting open area that would enhance the scope of the proposed model.

4. Considering each of the mental attitudes considered here separately rather than combining them to make an average factor is an important area of investigation. However, that may involve sorting out the issue of representation of the agent's reliability in terms of these attitudes, and that may require factoring out what sort of human errors could be induced in the presence of a particular mental attitude alone. For example, are the human errors due to stress and fear the same, or does stress give rise to other behaviour compared to fear.

5. Employing advanced similarity function would surely improve the performance and effectiveness of the retrieval mechanism.

Bibliography

Aamodt, A. & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, *7*(1), 39–59.

Agre, P. E. & Rosenschein, S. J. (Eds.). (2006). *Computational theories of interaction and agency* [Collected papers]. Cambridge, MA: MIT Press.

Alterman, R. (1988). Adaptive planning. *Cognitive Science*, *12*, 393–421.

Amaya, S. (2011). Slips. *Nous*, *47*(3), 559–576.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of Mind. *Psychological Review*, *111*(4), 1036–1060. Retrieved from http://act-r.psy.cmu.edu/papers/526/FSQUERY.pdf

Anderson, M. & McCartney, R. (2003). Diagram processing: Computing with diagrams. *Artificial Intelligence*, *145*(1-2), 181–226. doi:http://dx.doi.org/10.1016/S0004-3702(02)00383-1

Baddeley, A. D. (2007). *Working memory, thought and action*. Oxford: Oxford University Press.

Baddeley, A. D. (2010). Working memory. *Current Biology*, *20*(4), R136–R140.

Baddeley, A. D. & Hitch, G. (1974). Working memory. In G. H. Bower (Ed.), *Psychology of learning and motivation* (pp. 47–89). New York, NY: Academic Press.

Barlett, F. C. (1932). *Remembering: a study in experimental and social psychology*. Cambridge: Cambridge University Press.

Beer, R. D. (1995). A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, *72*(1–2), 173–215. doi:10.1016/0004-3702(94)00005-L

Bergmann, R., Althoff, K.-D., Breen, S., Göker, M., Manago, M., Traphöner, R., & Wess, S. (2003). Developing industrial case-based reasoning application: the INRECA methodology. In R. Bergmann, K.-D. Althoff, S. Breen, M. Göker, M. Manago, R. Traphöner, & S. Wess (Eds.), *Lecture notes in computer science*: *Vol. 1612. Developing industrial case-based reasoning applications* (pp. 1–216). Heidelberg, Germany: Springer-Verlag. doi:10.1007/b94998

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. West Sussex, England: John Wiley & Sons.

BP.com. (2010, September 8). Deepwater horizon accident investigation report [Web Posting]. Retrieved from http://www.bp.com/liveassets/bp_internet/globalbp/globalbp_uk_english/incident_response/STAGING/local_assets/downloads_pdfs/Deepwater_Horizon_Accident_Investigation_Report.pdf

Bratman, M. E. (1987). *Intentions, plans and practical reason*. Cambridge, MA: Harvard University Press.

Brooks, R., Arbel, T., & Precup, D. (2008). Anytime similarity measures for faster alignment. *Computer Vision and Image Understanding*, *110*(3), 378–389. doi:http://dx.doi.org/10.1016/j.cviu.2007.09.011

Buckland, M. (2005). *Programming game AI by example*. Sudbury, MA: Wordware Publishing Inc.

Busetta, P. [P.], Howden, N., Rönnquist, R., & Hodgson, A. (2000). Structuring BDI agents in functional clusters. In N. R. Jennings & Y. Lespérance (Eds.), *Lecture notes in computer science*: *Vol. 1757. Intelligent agents VI. Agent theories,*

architectures, and languages (pp. 277–289). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-540-74607-2_9

Busetta, P. [Paolo], Ronnquist, R., Hodgson, A., & Lucas, A. (1999). JACK intelligent agents — components for intelligent agents in java [Article]. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.6936&rep=rep1&type=pdf

Chen, T.-H., Wu, P.-H., & Chiou, Y.-C. (2004). An early fire-detection method based on image processing. In *ICIP*, October 24–27, 2004 (Vol. 3, pp. 1707–1710). Singapore.

Christou, M. & Myrto, K. A. (2012). *Safety of offshore oil and gas operations: Lessons from past accident analysis: Ensuring EU hydrocarbon supply through better control of major hazards*. Luxembourg: JRC Scientific and Policy Reports, Publications office of the European Union.

Danial, S. N., Khan, F., Veitch, B., & MacKinnon, S. N. (2013, July 4). *Dynamic learning and adapting: An AI based approach to enhance virtual training* [Research article]. See Chapter 3. Manuscript submitted for publication. Memorial University of Newfoundland, St. John's, NL, Canada.

Danziger, K. (2001). Sealing off the discipline: Wilhelm Wundt and the psychology of memory. In C. D. Green, M. Shore, & T. Teo (Eds.), *The transformation of psychology : influences of 19th century philosophy, technology, and natural science* (pp. 45–62). Washington, DC: American Psychological Association.

Dastani, M., van Riemsdijk, M. B., & Meyer, J.-J. C. (2005). Programming multiagent systems in 3APL. In R. H. Bordini, M. Dastani, J. Dix, & A. E. F. Seghrouchni (Eds.), *Multiagent programming: Languages, platforms, and applications* (pp. 39–67). Berlin, Germany: Springer-Verlag.

Duch, W. (2000). Similarity-based methods: a general framework for classification, approximation and association. *Control and Cybernetics*, *29*(4).

Fagin, R., Kumar, R., & Sivakumar, D. (2003). Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 acm sigmod international conference on management of data* (pp. 301–312). SIGMOD '03. San Diego, California: ACM. doi:10.1145/872757.872795

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison Wesley Longman Publishing Co., Inc.

Georgeff, M. P. & Ingrand, F. F. (1990, January). Real-time reasoning: The monitoring and control of spacecraft systems. In *Proc. sixth conference on artificial intelligence application*, May 5–9, 1990 (pp. 198–204). Santa Barbara, CA.

Georgeff, M. P. & Lansky, A. L. (1986, January). *A system for reasoning in dynamic domains: Fault diagnosis on the space shuttle* (Technical Note No. 375). Artificial Intelligence Center, SRI International, Menlo Park, CA.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1999). The Belief-Desire-Intention model of agency. In J. P. Muller, A. S. Rao, & M. P. Singh (Eds.), *Lecture notes in computer science*: *Vol. 1555. Intelligent agents v: agents theories, architectures, and languages* (pp. 1–10). Heidelberg, Germany: Springer-Verlag. doi:10.1007/3-540-49057-4_1

Hayes-Roth, B. (1995). An architecture for adaptive intelligent systems. *Artificial Intelligence*, *72*(1–2), 329–365. doi:10.1016/0004-3702(94)00004-K

Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J.-J. C. (1998). Formal semantics of an abstract agent programming language. In M. P. Singh, A. S. Rao, & M. J. Wooldridge (Eds.), *Lecture notes in computer science*: *Vol. 1365.*

Intelligent agents VI. Agent theories, architectures, and languages (pp. 215–229). Berlin, Germany: Springer-Verlag. doi:10.1007/BFb0026761

Ingrand, F. F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *7*(6), 34–44. doi:10.1109/64.180407

James, W. (1908). *Talks to teachers on psychology: And to students on some of life's ideals*. London, UK: Longmans, Green & Co.

Kemeny, J. G. (1979). *Report of the president's commission on the accident at the Three Mile Island*. Washington, DC.

Khakzad, N., Khan, F., & Amyotte, P. (2013). Quantitative risk analysis of offshore drilling operations: A bayesian approach. *Safety Science*, *57*, 108–117. doi:10.1016/j.ssci.2013.01.022

Khan, F., Sadiq, R., & Hussain, T. (2002). Risk-based process safety assessment and control measures design for offshore process facilities. *Journal of Hazardous Materials*, *94*(1), 1–36. doi:http://dx.doi.org/10.1016/S0304-3894(02)00004-3

Kim, A. (2008). Wilhelm Maximilian Wundt. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Fall 2008 ed.). Retrieved from http://plato.stanford.edu/archives/fall2008/entries/wilhelm-wundt/

Klein, G. (1998). *Sources of power: how people make decisions*. Cambridge, MA: MIT Press.

Knuth, D. E. (1969). *The art of computer programming: seminumerical algorithms*. Reading, MA: Addison Wesley Longman Publishing Co., Inc.

Kolodner, J. L. (1983). Reconstructive memory: A computer model. *Cognitive Science*, *7*(4), 281–328. doi:10.1016/S0364-0213(83)80002-0

Kruetzer, W. & McKenzie, B. (1991). *Programming for artificial intelligence: Methods, tools and application*. Reading, MA: Addison-Wesley.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1–64. doi:10.1016/0004-3702\(87\)90050-6

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*(1), 11–46. doi:10.1023/A:1022639103969

Lin, C. J., Yenn, T.-C., & Yangb, C.-W. (2010). Optimizing human-system interface automation design based on a skill-rule-knowledge framework. *Nuclear Engineering and Design*, *240*(7), 1897–1905.

Lincoln, N. K. & Veres, S. M. (2013). Natural language programming of complex robotic BDI agents. *Journal of Intelligent & Robotic Systems*, *71*(2), 211–230.

Maes, P. (1995). Artificial life meets entertainment: Lifelike autonomous agents. *Communications of the ACM*, *38*(11), 108–114. doi:10.1145/219717.219808

Majumdar, P. & Samanta, S. K. (2013). Decision making based on similarity measure of vague soft sets. *Journal of Intelligent & Fuzzy Systems*, *24*, 637–646.

Matsumoto, M. & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation - Special issue on uniform random number generation*, *8*(1), 3–30. doi:10.1145/272991.272995

McCann, H. J. (1991). *Noûs*, *25*(2), 230–233. Retrieved from http://www.jstor.org/stable/2215590

McCarthy, J. (1979, February). *Ascribing mental qualities to machines* (Memo No. 326). Stanford AI Lab, Stanford University, CA: Author.

McVee, M. B., Dunsmore, K., & Gavelek, J. R. (2005). Schema theory revisited. *Review of Educational Research*, *75*(4), 531–566.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior*. New York: Holt.

Millington, I. & Funge, J. (2009). *Artificial intelligence for games* (2nd ed.). Burlington, MA: Morgan Kaufmann Publishers.

Minsky, M. L. (1974, June 1). *A framework for representing knowledge* (AI Memos No. 1959-2004). MIT AI Labs, Massachusetts Institute of Technology, MA: Author. Reprinted in P. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 1975. Retrieved from http://dspace.mit.edu/handle/1721.1/6089

Murch, R. & Johnson, T. (1999). *Intelligent software agents*. Upper Saddle River, NJ: Prentice-Hall PTR.

Musharraf, M., Hussain, J., Khan, F., Veitch, B., MacKinnon, S., & Imtiaz, S. (2013). Human reliability assessment during offshore emergency conditions. *Safety Science*, *59*, 19–27. doi:10.1016/j.ssci.2013.04.001

Musharraf, M., Khan, F., Veitch, B., MacKinnon, S. N., & Imtiaz, S. (2013). Human factor risk assessment during emergency condition in harsh environment. In *Proc. of the 32nd international conference on ocean, offshore and Arctic engineering (OMAE), June 9-14, 2013, Nantes, France*, June 9–14, 2013. Nantes, France.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A. (1992a). Précis of unified theories of cognition. *Behavioural and Brain Sciences*, *15*(3), 425–437. doi:10.1017/S0140525X00069478

Newell, A. (1992b). Unified theories of cognition and the role of Soar. In J. A. Michon & A. Akyürek (Eds.), *Soar: A cognitive architecture in perspective* (Vol. 10, pp. 25–79). Studies in Cognitive Systems. Cambridge, MA: Springer Nether-

lands, Copyright of Kluwer Academic Publishers. doi:10.1007/978-94-011-2426-3_3

Newell, A. & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

NFPA.org. (2013, August 12). NFPA 10: Standard for portable fire extinguishers [Online]. Retrieved from http://www.nfpa.org

Norman, D. A. (2002). *The design of everyday things*. NY: Doubleday.

Park, J. L. & Band, W. (1976). Mutually exclusive and exhaustive quantum states. *Foundations of Physics*, *6*(2), 157–172.

Phung, T., Winikoff, M., & Padgham, L. (2005). Learning within the BDI framework: An empirical analysis. In R. Khosla, R. J. Howlett, & L. C. Jain (Eds.), *Lecture notes in computer science*: *Vol. 3683. Knowledge-based intelligent information and engineering systems* (pp. 282–288). Heidelberg, Germany: Springer-Verlag. doi:10.1007/11553939_41

Rabin, S. (2002). *AI game programming wisdom*. Rockland, MA: Charles River Media, Inc.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. *Proceedings of Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Agents breaking away: Agents breaking away (MAAMAW-96)*, 42–55.

Rao, A. S. & Georgeff, M. P. (1992, October). An abstract architecture for rational agents. In B. Nebel, C. Rich, & W. Swartout (Eds.), *Proceedings of the third international conference (KR '92), October 25-29, 1992*. Principles of Knowledge Representation and Reasoning. Cambridge, MA: Morgan Kaufmann.

Rao, A. S. & Georgeff, M. P. (1995). BDI agents: from theory to practice. In L. Gasser & V. Lesser (Eds.), *Proceedings of the first international conference on multi-agent systems*: (pp. 312–319). San Francisco, CA.

Rasmussen, J. & Jensen, A. (1974). Mental procedures in real-life tasks: a case study of electronic troubleshooting. *Ergonomics*, *17*, 293–307.

Reason, J. (1990). *Human error*. Cambridge, UK: Cambridge University Press.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In *Proceedings of game developers conference 1999 san jose* (pp. 763–782). San Francisco, CA: Miller Freeman Game Group.

Rickel, J. & Johnson, W. L. (1998, May). STEVE: A pedagogical agent for virtual reality. In *Proc. of the 2nd international conference autonomous agents, May, 1998, Minneapolis-St. Paul, MN*. Minneapolis - St. Paul, MN: ACM Press.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1992). *The Soar papers: Research on integrated intelligence*. Cambridge, MA: MIT Press.

Ross, S. A. (1973). The economic theory of agency: the principal's problem. *The American Economic Review*, *63*(2), 134–139.

Rumelhart, D. E. (1975). Notes on a schema for stories. In D. Borrow & A. Collins (Eds.), *Representation and understanding: studies in cognitive science*. New York: Academic Press.

Rumelhart, D. E. & Ortony, A. (1977). The representation of knowledge in memory. In R. C. Anderson, R. J. Shapiro, & W. E. Montague (Eds.), *Schooling and the acquisition of knowledge*. Hillsdala, NJ: Erlbaum.

Russel, S. J. & Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *2*, 575–609. doi:10.1613/jair.133

Russell, S. J. & Norvig, P. (1995). *Artificial intelligence: a modern approach*. Englewood Cliffs, NJ: Prentice-Hall.

Santini, S. & Jain, R. (1996). Similarity matching. In S. Z. Li, D. P. Mital, E. K. Teoh, & H. Wang (Eds.), *Proceedings of second Asian conference on computer vision, ACCV '95 Singapore, December 5-8, 1995. Invited session papers, recent development in computer vision* (Vol. 1035, pp. 570–580). Lecture Notes in Computer Science. Heidelberg: Springer-Verlag.

Schalkoff, R. J. (2011). *Intelligent systems. Principles, paradigms, and pragmatics*. Sudbury, MA: Jones and Bartlett Publishers.

Scheerer, E. (1980). Wilhelm Wundt's psychology of memory. *Psychological Research*, *42*, 135–155.

Schleiffer, R. (2005). An intelligent agent model. *European Journal of Operational Research*, *166*(3), 666–693. doi:10.1016/j.ejor.2004.03.039

Schmidt, R. A. (1975). A schema theory of discrete motor skill learning. *Psychological Review*, *82*, 225–260.

Seel, N. (1989). *Agent theories and architectures* (Doctoral thesis, Surrey University, Guildford, UK).

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–91. doi:10.1016/0004-3702(93)90034-9

Skogdalen, J. E., Khorsandi, J., & Vinnem, J. E. (2012). Evacuation, escape, and rescue experiences from offshore accidents including the deepwater horizon. *Journal of Loss Prevention in the Process Industries*, *25*(1), 148–158. doi:10.1016/j.jlp.2011.08.005

Sloman, A. (1995). Musings on the roles of logical and non-logical representations in intelligence. In J. Glasgow, N. Narayanan, & B. Chandrasekaran (Eds.), *Diagrammatic reasoning*: *Cognitive and computational perspectives*. Cambridge, MA: MIT Press.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., & Jain, R. (2000). Content based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(12), 1349–1380. doi:http://dx.doi.org/10.1109/34.895972

Stentiford, F. (2007). Attention-base similarity. *Pattern Recognition*, *40*(3), 771–783. doi:http://dx.doi.org/10.1016/j.patcog.2006.05.014

Sterling, L. S. & Taveter, K. (2009). *The art of agent-oriented modeling*. Cambridge, MA: MIT Press.

Sullivan, L. E. (2009, December). *The SAGE glossary of the social and behavioral sciences*. Thousand Oaks, California: SAGE Publications Inc. doi:10.4135/9781412972024

Swain, A. D. & Guttmann, H. E. (1983). *Handbook of human reliability analysis with emphasis on nuclear power: final report*. Washington: US Nuclear Regulatory Commission. Retrieved from http://books.google.ca/books?id=BCG7QwAACAAJ

Taatgen, N. A., Lebiere, C., & Anderson, J. R. (2006). Modelling paradigms in ACT-R. In R. Sun (Ed.), *Cognition and multi-agent interaction: From cognitive modeling to social simulation* (pp. 29–52). Cambridge, UK: Cambridge University Press. Retrieved from http://www.ai.rug.nl/~niels/publications/taatgenLebiereAnderson.pdf

Tate, A. (2003, August). <I-N-C-A>: A shared model for mixed-initiative synthesis tasks. In G. Tecuci (Ed.), *Proceedings of the workshop on mixed-initiative intelligent systems (MIIS) at the eighteenth international joint conference on artificial intelligence (IJCAI-03), 9 August 2003* (pp. 125–130). Acapulco, Mexico.

Tompkins, O. (2010). Panic attacks. *AAOHN Journal*, *58*(6), 268. doi:10.3928/08910162-20100526-07

Tribus, M. (1961). *Thermostatics and thermodynamics: An introduction to energy, information and states of matter, with engineering applications*. New York: Van Nostrand.

Tversky, A. (1977). Features of similarity. *Psychological Review*, *84*(4), 327–352. doi:http://dx.doi.org/10.1037/0033-295X.84.4.327

Velleman, J. D. (1991, April 1). *The Philosophical Review*, *100*(2), 277–284. Retrieved from http://www.jstor.org/stable/2185304

Verstockt, S., Hoecke, S. V., Tilley, N., Merci, B., Sette, B., Lambert, P., ... Walle, R. V. (2011). Firecube: a multi-view localization framework for 3d fire analysis. *Fire Safety Journal*, *46*(5), 262–275. doi:10.1016/j.firesaf.2011.03.001

Verstockt, S., Vanoosthuyse, A., Hoecke, S. V., Lambert, P., & Walle, R. V. (2010). Multi-sensor fire detection by fusing visual and non-visual flame features. In A. Elmoataz, O. Lezoray, F. Nouboud, D. Mammass, & J. Meunier (Eds.), *Proceedings of the 4th international conference, ICISP 2010, Trois-rivières, QC, Canada, June 30-July 2, 2010, image and signal processing* (Vol. 6134, pp. 333–341). Lecture Notes in Computer Science. Heidelberg: Springer-Verlag.

Wang, J., Ju, S.-E., & Liu, C.-N. (2006). Agent-oriented prababilistic logic programming. *Journal of Computer Science & Technology*, *21*(3), 412–417.

Watanabe, S. (1969). *Knowing and guessing: a formal and quantitative study*. NY: John Wiley & Sons, Inc.

WAtoday.com.au. (2009, August 29). Oil spill off WA coast much larger than reported: greens [Online Newspaper]. Retrieved from http://www.watoday.com.au/wa-news/oil-spill-off-wa-coast-much-larger-than-reported-greens-20090828-f2i7.html

Wickler, G., Potter, S., Tate, A., Pechoucek, M., & Semsch, E. (2007, November). Planning and choosing: Augmenting HTN-based agents with mental attitudes.

In *International conference on intelligent agent technology (IAT 2007), Silicon Valey, 2-5 November 2007* (pp. 222–228). 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. Fremont, CA: IEEE Computer Society. doi:10.1109/IAT.2007.64

Wooldridge, M. J. & Jennings, N. R. (1995). Agent theories, architectures and languages: a survey. In M. J. Wooldridge & N. R. Jennings (Eds.), *Lecture notes in computer science*: *Vol. 890. Intelligent agents. proceedings ofECAI-94 workshop on agent theories, architectures, and languages amsterdam, the netherlands, august 8-9, 1994* (pp. 1–39). Heidelberg, Germany: Springer-Verlag. doi:10.1007/3-540-58855-8

Wundt, W. (1905). *Grundriss der psychologie*. Leipzig: Engelmann.

Yan, W., Zanni-Merk, C., Rousselot, F., & Cavallucci, D. (2013). Ontology matching for facilitating inventive design based on semantic similarity and case-based reasoning. *International Journal of Knowledge-based and Intelligent Engineering Systems*, *17*, 243–256.

Yang, S.-Y. (2013). Developing an energy-saving and case-based reasoning information agent with web service and ontology techniques. *Expert Systems with Applications*, *40*(9), 3351–3369.

Appendices

Appendix A

The C++ class definition of the agent-model

The following code describes only the main points in the implementation of the model depicted in Figure 3.1. The complete implementation is available with the source code provided in the enclosed CD.

```
#ifndef AGENT_H
#define AGENT_H
//header declarations omitted to save space, please see the
   accompanying CD.
class Agent : public MovingEntity
{
private:
     enum Status{alive, dead, spawning};
     enum GoalOutcome{increment, decrement, negate, noeffect};
public:
     struct CurrentDecisionRecord{
          int m_nTypeofToolUsed;
          bool WasInterrupted;
          int m_nHazardLevel;
          //the entry which was there in the HashTable
          KBunit m_KBu;
          //the calling condition of KBunit in the KB
          CCond m_ccond;
          Vector2D m_vPos;//position of hazard
          void Clear(){
               m_vPos=Vector2D(-100,-100);
               m_nHazardLevel=-1; m_KBu.fp=0;
               m_KBu.sol=KnowledgeBaseUnit::no_record;
               m_ccond.Clear(); WasInterrupted=false;
          }
     };
private:
     typedef struct CurrentFireReactionRecord{
          std::vector<Vector2D> m_vecTemp;
          std::vector<Vector2D> m_vecFlameLength;
```

```
            std::vector<Vector2D> m_vecFlameWidth;
            int index;
            CurrentFireReactionRecord(){index=0;}
            void ClearIndex(){index=0;}
            void Clear(){
                  m_vecTemp.clear();
                  m_vecFlameLength.clear();
                  m_vecFlameWidth.clear();
                  index = 0;
            }
            void save(double t,double fl, double fh){
                  m_vecTemp.push_back(Vector2D((double)index, t));
                  m_vecFlameLength.push_back(Vector2D((double)index,
                        fl));
                  m_vecFlameWidth.push_back(Vector2D((double)index,
                        fh));
                  index+=1;
            }
      }CFRR;
public:
      CurrentDecisionRecord* m_pCDR;
      CFRR* m_pCfrr;
      GoalOutcome goalloutcome;
      int m_nIgnoreFireClassInfo;
private:
      Vector2D m_vFacing;
      //an agent only perceives other agents or obstacles within this
         field of view
      double m_dFieldOfView;
      bool m_bLockRecognizing;
      bool m_bLockEvaluateScenarioMethod;
      bool m_bLockPercieve;
      bool m_bLockMakeCallingConditionMethod;
      //a measure of surprise will be stored here
      double m_dSurprise;
      double m_dIgnorance;
      double m_Min_Ign;
      double m_Max_ign;
      //minimum if value is 0; maximum if value is 1
      unsigned int m_nIgnoranceFlag;
      bool m_bAlarm;
      //1 is Process and 2 is Muster
      unsigned int m_nAlarmType;
      bool m_bAlarmRemembered;
      bool m_bIsCarryingExtinguisher;
      //default value is false, true means to send alarm message
      bool m_bSendAlarmMessageToOtherAgents;
      //to be used in deciding if to use exact matching or similarity-
         based
      double m_dMinProbToShowStress;
      unsigned int m_nKBFileNamePrefix;
```

```
    //although its value is equal to m_vHazRecord.size() but I'm
       storing it sepearately also
    int m_nNumHazardSeenCurrently;
    int m_nTotalHazLevel;
    //Level of Currently observed hazard
    int m_nCurrHazLevel;
    //the probability value above which a hazard is considered
    double m_dThresholdProb;
    //a knowledgebase object
    KB* m_pKB;
    //human factor object
    HumanFactor m_HFactor;
    //container for all probability of selection of each Sols.
    std::vector<double> m_vProbabilities;
    ExtinguishingToolsRecord m_ExtToolRec;
    //it is the database of info comming from Fire. Its the inital
       unprocessed info about hazard
    MapExt<Vector2D,ImgData> m_MapExtImgDataLocalCopy;
    //it contains complete rec of fire info after processing in
       Recognition Phase
    MapExt<Vector2D,PartialHazardRecord> m_MapExtHazRecord;
    //a list of hazard objects of which the surroundings have been
       calculated already, thus surroundings need to update only
    vector<Vector2D> m_vPercieved;
    //its the bit field for Surrounding ID. The first argument is
       the position of hazard object
    MapExt<Vector2D, CCond> m_MapSurIds;
    std::vector<KBunit> m_vtheCorrectSol;
    //I'll put those events (hazards) which are logically evaluated,
        so that they don't get evaluated again.
    std::set<Vector2D> SetOfCheckedEvents;
    WorkingMemory* m_pWM;
    Agent_Steering* m_pSteering;
    //the currently observed hazard
    Vector2D m_pCurrHazardPos;
    //a vertex buffer containing the agent's geometry
    std::vector<Vector2D> m_vecAgentVB;
    //the buffer for the transformed vertices
    std::vector<Vector2D> m_vecAgentVBTrans;
    //alive, dead or spawning?
    Status m_Status;
    //a pointer to the world data
    Environment* m_pWorld;
    //this object handles the arbitration and processing of high
       level goals
    Goal_Router* m_pRouter;
    //this is a class that acts as the agent sensory memory.
    Agent_SensoryMemory* m_pSensoryMem;
    //the agent uses this to plan paths
    Agent_PathPlanner* m_pPathPlanner;
    //this is responsible for choosing the agent's current target
    Agent_TargetingSystem* m_pTargSys;
```

```
    //A regulator object limits the update frequency of a specific
       AI component
    Regulator* m_pGoalArbitrationRegulator;
public:
    Agent(Environment* world, Vector2D pos,unsigned int kbfilename);
    virtual ~Agent();
    void UnlockEvaluateScenarioNaturallyMethod(){
        m_bLockEvaluateScenarioMethod=false;
    }
    void UnlockPercieve(){m_bLockPercieve=false;}
    MapExt<Vector2D,ImgData>& GetMapExtImgDataLocalCopy(){
        return m_MapExtImgDataLocalCopy;
    }
    MapExt<Vector2D,PartialHazardRecord>& GetMapExtHazRecord(){
        return m_MapExtHazRecord;
    }
    std::vector<double>& GetProbabilities(){
        return m_vProbabilities;
    }
    void ClearCheckedEvent(const Vector2D&);
    //clears calling condition flag from m_MapExtHazRecord so that
       another calling condition could be generated on next visit
       of the same hazard
    void ClearCallingConditionFlag(const Vector2D&);
    void ClearAllRecords();
    void SetGoalOutcome(unsigned int outcome){
        goaloutcome=(GoalOutcome)(outcome%4);
    }
    void SetExtToolRec(FireExtinguishers* pExt,Vector2D Pos){
        m_bIsCarryingExtinguisher=true;
        m_ExtToolRec.iniPos=Pos;
        m_ExtToolRec.pExtTool=pExt;
    }
    FireExtinguishers* GetGrabbedExtRef()const {
        return m_ExtToolRec.pExtTool;
    }
    Vector2D GetGrabbedExtPos()const{return m_ExtToolRec.iniPos;}
    bool isCarryingExtTool()const{return m_bIsCarryingExtinguisher;}
    //called only from EvaluateScenarioNaturally()
    void AddSol4Going2MusterInCaseOfManyHazards();
    bool ScanForHazards();
    void MakeCallingCondition();
    void RecognizeHazard();
    void PercieveSurroundings();
    //Find similar and frequent KB-Units
    void EvaulateScenarioLogically();
    //Pick one KB-Unit on the basic of reliability or human factors
    void EvaluateScenarioNaturally(double CurrentTime);
    //to save the final decision
    void SaveDecision(const Vector2D Position_of_hazard);
    //if heard alarm or not
    bool IsAlarm()const{return m_bAlarm;}
```

```
    bool IsAlarmRemembered()const{return m_bAlarmRemembered;}
    void AlarmRemembered(){m_bAlarmRemembered=true;}
    void AlarmForgot(){m_bAlarmRemembered=false;}
    //this records fire reaction when agent attempt to kill it
    void RecordFireReaction(const Fire* pFire,
         int TypeOfExtinguisher);
    unsigned int GetAlarmType()const{return m_nAlarmType;}
    void SetAlarmType(unsigned int alarm_type){
         m_bAlarm=true;
         m_nAlarmType=alarm_type;
    }
    void ClearAlarm(){m_bAlarm=false;m_nAlarmType=0;}
    //Measure Surprise
    double MeasureSurprise(int);
    //Measure Ignorance or expected surprise
    void MeasureIgnorance();
    double MaxIgnorance();
    double MinIgnorance();
    double ScaleIgnorance(double val);
    double ScaleSurprise(double val);
    double GetIgnorance()const {return m_dIgnorance;}
    double GetSurprise()const {return m_dSurprise;}
    //this assess the recorded reaction as to +ve or -ve decision
    void AssessFireReaction(const Fire* pFire,
         int TypeOfExtinguisher);
    //pS = value of p in HumanFactor object
    void GenProbabilities(int n,double pS);
    HumanFactor& GetHumanFactor(){return m_HFactor;}
    bool IgnoreFireClass();
    MapExt<Vector2D,ImgData>& GetMapExt(){
         return m_MapExtImgDataLocalCopy;
    }
    int getTotalHazLevel()const {return m_nTotalHazLevel;}
    void ClearReflexAction(){m_HFactor.ClearReflexAction();}
    WorkingMemory* getWM()const {return m_pWM;}
    MapExt<Vector2D, CCond>& GetSurID(){return m_MapSurIds;}
    KB* GetKB()const{return m_pKB;}
    Environment* const GetWorld(){return m_pWorld;}
    Agent_Steering* const GetSteering(){return m_pSteering;}
    Agent_PathPlanner* const GetPathPlanner(){
         return m_pPathPlanner;
    }
    Goal_Router* const  GetRouter(){return m_pRouter;}
    bool isAtPosition(Vector2D pos)const;
    //file objects for storing results
    ofstream fresults, fresults2;
    void UpdateKB();
    void Render();
    void Update();
    bool HandleMessage(const Telegram& msg);
};
#endif
```

Appendix B

The knowledgebase and related data structures

```
#ifndef KNOWLEDGEBASE_H
#define KNOWLEDGEBASE_H

typedef struct KnowledgeBaseUnit
{
private:
     //serialization support
     friend class boost::serialization::access;
     template <class Archive>
     void serialize(Archive& arc, const unsigned int version)
     {    arc & sol & fp;     }
public:
     enum solution_space
     {
          no_record=99, water_based, foam_based, co2_based,
          dry_chemical_based, purple_K_dry_chemical, dry_powder,
          multipurpose_dry_chemical, wet_chemical, sand_buckets,
          go_muster_station, leave_everything_and_be_atentive,
          proceedcurrentgoal
     };
     solution_space sol;
     //frequency of bias
     int fp;
     //some operator overloads
     friend bool operator == (const KnowledgeBaseUnit& c1,
          const KnowledgeBaseUnit& c2);
     friend bool operator != (const KnowledgeBaseUnit& c1,
          const KnowledgeBaseUnit& c2);
     friend std::ostream & operator<<(std::ostream &os,
          const KnowledgeBaseUnit &kbunit);
     bool operator < (const KnowledgeBaseUnit kbu) const{
          return sol<kbu.sol;
     }
```

```
     //nested class to facilitate customized sorting
     class greater
     {
          public:
               bool operator()(const KnowledgeBaseUnit& val1,
                    const KnowledgeBaseUnit& val2){
                         return val1.fp>val2.fp;
               }
     };
     //ctor
     KnowledgeBaseUnit(){
          fp=0;
          sol=no_record;
     }
     std::string ToString() const;
}KBunit, SOLUTION_SPACE;

//data structure to save the count of similar bits
typedef struct SimilarBitsCount
{
     std::multimap<CCond, KBunit,CCond::LessThan>::iterator m_It;
     int m_nCount;
     SimilarBitsCount(std::multimap<CCond, KBunit,CCond::LessThan>::
        iterator it, int cnt)
          :m_It(it),m_nCount(cnt){}
     class greater
     {
     public:
          bool operator()(const SimilarBitsCount& val1, const
             SimilarBitsCount& val2)
          {
               if(val1.m_nCount==val2.m_nCount)
               {
                    if(val1.m_It->second.fp>val2.m_It->second.fp)
                         return true;
                    else
                         return false;
               }
               else if(val1.m_nCount>val2.m_nCount)
                    return true;
               else
                    return false;
          }
     };
}SimilarityCount, SimilarityMatchingDataStructure;
```

```
//The Knowledge base class
typedef class Knowledgebase
{
public:
     enum UpdateParams{increment,decrement,negate,noeffect};
private:
     //serialization support
     friend class boost::serialization::access;
     template <class Archive>
     void serialize(Archive& arc, const unsigned int version){
          arc & m_MMap;
     }
    friend std::ostream & operator<<(std::ostream &os, const
       Knowledgebase &kb);
     //reads from an .ini file
     bool Read();
public:
     Knowledgebase(Agent* pOwner=NULL);
     size_t Size()const{return m_MMap.size();}
     //loads the previously saved boost text archive object of m_MMap
         for this agent or call Read() to read from the .ini file
     bool Load();
     bool Save();
     void SetPath(std::string path){    m_strPath=path;}
     bool Find(const CCond&, std::vector<KBunit>& vtheCorrectSol);
     //the method that implements similarity-matching and frequency
        gambling techniques
     void
     FindBasedOnSimilarityMatchingAndFrequencyBias(const CCond& rcnd,
          std::vector<KBunit>& refVecOfSols);
     //the method to update the KB
     void Update(const CCond&, const KBunit&, unsigned int bParam);
     void Write(string filename);
private:
     //the data structure to store KB units with CCond mapped
     std::multimap<CCond, KBunit,CCond::LessThan> m_MMap;
     typedef std::multimap<CCond, KBunit,CCond::LessThan>::
          iterator m_MMapIter;
     double m_dLastUpdateTime;
     m_MMapIter m_itKB;
     //the owner of this KB
     Agent* m_pOwner;
}KB;
//A data structure to save the final retreived KB-Units
typedef class ConsiousMemory
{
public:
     unordered_map<Vector2D, vector<KBunit>> m_HashTable;
     friend ostream& operator<<(ostream& out, const ConsiousMemory&);
}WorkingMemory, WM;
#endif
```

Appendix C

The Goal_Router class

```
#ifndef GOAL_ROUTER_H
#define GOAL_ROUTER_H
//headers omitted to save space
class Goal_Router : public Goal_Composite<Agent>
{
public:
     Goal_Router(Agent* pBot);
     ~Goal_Router();
     //this method selects the one best goal
     void Router();
     //returns true if the given goal is not at the front of the
        subgoal list
     bool notPresent(unsigned int GoalType)const;
     //the usual methods for goal oriented models
     int  Process();
     void Activate();
     void Terminate(){}
     //top level goal types
     void AddGoal_MoveToPosition(Vector2D pos);
     void AddGoal_GetItem(unsigned int ItemType);
     void AddGoal_Explore();
     void AddGoal_AttackTarget();
     void AddGoal_GoMusterStation();
     void AddGoal_UseWaterBasedExtinguisher();
     void AddGoal_UseDryChemBasedExtinguisher();
     void AddGoal_UseCFCBasedExtinguisher();
     void AddGoal_UseCO2BasedExtinguisher();
     void AddGoal_UseFoamBasedExtinguisher();
     void AddGoal_UseDryPowderBasedExtinguisher();
     void AddGoal_UsePurpleKDryChemicalExtinguisher();
     void AddGoal_UseWetChemicalExtinguisher();
     void AddGoal_UseMultiPurpDryChemicalExtinguisher();
     void AddGoal_LeaveEverythingAndBeAttentive();
     void AddGoal_ProceedCurrentGoal();
     void QueueGoal_MoveToPosition(Vector2D pos);
};
#endif
```

Appendix D

Some important methods

```
void Goal_Router::Router()
{
     m_pOwner->RecognizeHazard();
     if(m_pOwner->GetNumHazardsSeenCurrently()>0
          ||m_pOwner->IsAlarm())
     {
          GetNumHazCurrently++;
          m_pOwner->GetHumanFactor().Update(m_pOwner);
          m_pOwner->PercieveSurroundings();
          m_pOwner->MakeCallingCondition();
          m_pOwner->EvaulateScenarioLogically();
          m_pOwner->
               EvaluateScenarioNaturally(Clock->GetCurrentTime());
     }
     std::unordered_map<Vector2D, vector<KBunit>>::
          const_iterator iH = m_pOwner->getWM()->m_HashTable.cbegin()
             ;
     if(iH!=m_pOwner->getWM()->m_HashTable.cend()
          && iH->second.size()>0)
     {
          switch(iH->second[0].sol)
          {
               case KBunit::leave_everything_and_be_atentive:
                    m_pOwner->GetRouter()
                         ->AddGoal_LeaveEverythingAndBeAttentive();
                    break;
               case KBunit::go_muster_station:
                    m_pOwner->GetRouter()->AddGoal_GoMusterStation();
                    break;
               case KBunit::water_based:
                    m_pOwner->GetRouter()->
                         AddGoal_UseWaterBasedExtinguisher();
                    break;
               case KBunit::co2_based:
                    m_pOwner->GetRouter()->
                         AddGoal_UseCO2BasedExtinguisher();
                    break;
```

```
                case KBunit::dry_chemical_based:
                     m_pOwner->GetRouter()->
                          AddGoal_UseDryChemBasedExtinguisher();
                     break;
                case KBunit::foam_based:
                     m_pOwner->GetRouter()->
                          AddGoal_UseFoamBasedExtinguisher();
                     break;
                case KBunit::dry_powder:
                     m_pOwner->GetRouter()->
                          AddGoal_UseDryPowderBasedExtinguisher();
                     break;
                case KBunit::purple_K_dry_chemical:
                     m_pOwner->GetRouter()->
                          AddGoal_UsePurpleKDryChemicalExtinguisher();
                     break;
                case KBunit::multipurpose_dry_chemical:
                     m_pOwner->GetRouter()->
                          AddGoal_UseMultiPurpDryChemicalExtinguisher
                             ();
                     break;
                case KBunit::wet_chemical:
                     m_pOwner->GetRouter()->
                          AddGoal_UseWetChemicalExtinguisher();
                     break;
                case KBunit::proceedcurrentgoal:
                     m_pOwner->GetRouter()->
                          AddGoal_ProceedCurrentGoal();
                     break;
                default:
                     m_pOwner->GetRouter()->AddGoal_Explore();
                     break;
           };
     }
     else
           m_pOwner->GetRouter()->AddGoal_Explore();
}
```

```
//Implementation of Algorithm 1
void Agent::RetrieveKBUnits()
{    //temporary vector for KB-Units
     std::vector<KBunit> VecOfSols;
     std::set<Vector2D>::iterator itSet;
     for(std::map<Vector2D, CCond>::
          iterator it=m_MapSurIds.m.begin();
               it!=m_MapSurIds.m.end(); ++it)
     {
          itSet=SetOfCheckedEvents.find(it->first);
          if(itSet==SetOfCheckedEvents.end())
          {
               //m_vVecOfSols is an ordered vector, first by
                  Similarity then by frequency of bias. The code in
                  the following if block attempts at exact matching
                  in case if agent is not showing stressed and also
                  if there are exact matches in KB otherwise go for
                  bringing similar KB-units
               if(! (m_pKB->Find(it->second, VecOfSols)
                    &&(m_HFactor.getProbability()>
                       m_dMinProbToShowStress)))
               {
                    m_pKB->
                       FindBasedOnSimilarityMatchingAndFrequencyBias
                       (it->second,VecOfSols);
               }
               for(std::vector<KBunit>::
                    iterator itVec=VecOfSols.begin();
                    itVec!=VecOfSols.end();++itVec)
               m_pWM->m_HashTable[it->first].push_back(*itVec);
               VecOfSols.clear();
               //in order to use it again in the loop.
               m_vtheCorrectSol.clear();
               //tag this event that it is now checked
               SetOfCheckedEvents.insert(it->first);
          }
     }
}

//Exact matching based on multimap's find method
bool KB::Find(const CCond& rcnd, std::vector<KBunit>& vtheCorrectSol)
{
     std::pair<std::multimap<CCond,KBunit,CCond::LessThan>::iterator,
          std::multimap<CCond,KBunit,CCond::LessThan>::iterator>
             RangeOfEqualSols;
     RangeOfEqualSols = m_MMap.equal_range(rcnd);
     for (std::multimap<CCond,KBunit,CCond::LessThan>::iterator it=
        RangeOfEqualSols.first;
               it!=RangeOfEqualSols.second; ++it)
          vtheCorrectSol.push_back(it->second);
     if(vtheCorrectSol.size()>0)
```

```
            std::sort(vtheCorrectSol.begin(),vtheCorrectSol.end(),
                KBunit::greater());
        //remove all those with -ve frequencies, because they are wrong
        vector<KBunit>::iterator it;
        for(int i=0;i<vtheCorrectSol.size();++i){
            if(vtheCorrectSol[i].fp<0){
                vtheCorrectSol.erase(i+vtheCorrectSol.begin());
                i--;
            }
        }
        return (vtheCorrectSol.size()>0)?true:false;
}
//Supporting method
bool KB::FindNegative(const CCond& rcnd, std::vector<KBunit>&
    vtheCorrectSol)
{
        std::pair<std::multimap<CCond,KBunit,CCond::LessThan>::iterator,
            std::multimap<CCond,KBunit,CCond::LessThan>::iterator>
                RangeOfEqualSols;
        RangeOfEqualSols = m_MMap.equal_range(rcnd);
        for (std::multimap<CCond,KBunit,CCond::LessThan>::iterator it=
            RangeOfEqualSols.first;
                it!=RangeOfEqualSols.second; ++it)
            vtheCorrectSol.push_back(it->second);
        if(vtheCorrectSol.size()>0)    std::sort(vtheCorrectSol.begin(),
            vtheCorrectSol.end(),KBunit::greater());
        //remove all those with +ve frequencies, because they are right
        vector<KBunit>::iterator it;
        for(int i=0;i<vtheCorrectSol.size();++i){
            if(vtheCorrectSol[i].fp>=0){
                vtheCorrectSol.erase(i+vtheCorrectSol.begin());
                i--;
            }
        }
        return (vtheCorrectSol.size()>0)?true:false;
}
//Main method in Algorithm 1, Hamming distance based similarity-
    matching and frequency bias based retrieval
void KB::FindBasedOnSimilarityMatchingAndFrequencyBias(const CCond&
    rcnd, std::vector<KBunit>& refVecOfSols)
{
        double percentsimilarity=0.0;
        m_itKB=m_MMap.begin();
        std::vector<KBunit> vtheSols,negvtheSols;
        KBunit theSol;
        std::vector<unsigned int> CountBits;
        unsigned int CountBitsCurr=0;//each
        std::vector<SimilarityCount> SimilarityVec;
        std::vector<SimilarityCount>::iterator itSim;
        bool nofire= !rcnd.isFire();
        bool dontuse=false;
        FindNegative(rcnd,negvtheSols);
```

```
double simparam=script->GetDouble("PercentSimilarityParam");
if(rcnd.isFire())
{
     for(m_itKB=m_MMap.begin(); m_itKB!=m_MMap.end();++m_itKB)
     {
          if(m_itKB->first.isFire())
          {
               CountBitsCurr++;//due to matching hazards
               if(rcnd.isElectricalLines()==m_itKB->first.
                  isElectricalLines())
                    CountBitsCurr++;
               if(rcnd.isSmoke()==m_itKB->first.isSmoke())
                    CountBitsCurr++;
               if(rcnd.isFuelCarryingLines()==m_itKB->first.
                  isFuelCarryingLines())
                    CountBitsCurr++;
               if(rcnd.isMetalDust()==m_itKB->first.isMetalDust
                  ())
                    CountBitsCurr++;
               if(rcnd.isOrdinaryCombustible()==m_itKB->first.
                  isOrdinaryCombustible())
                    CountBitsCurr++;
               if(rcnd.isUnkownSurrounding()==m_itKB->first.
                  isUnkownSurrounding())
                    CountBitsCurr++;
               if(rcnd.GetHazardLevel()==m_itKB->first.
                  GetHazardLevel())
                    CountBitsCurr+=3;
               if(!m_pOwner->IgnoreFireClass())
                    if(rcnd.GetFireClass()==m_itKB->first.
                       GetFireClass())
                         CountBitsCurr+=3;
               for(int i=0;i<negvtheSols.size();++i){
                    if(negvtheSols[i].sol==m_itKB->second.sol){
                         dontuse=true;
                         break;
                    }
                    dontuse=false;
               }
               if(!dontuse){
                    percentsimilarity=(double)CountBitsCurr/(
                       double)rcnd.GetSIDlen()*100.0;
                    if(percentsimilarity>=simparam)
                         SimilarityVec.push_back(
                            SimilarBitsCount(m_itKB,
                            CountBitsCurr));
               }
               CountBitsCurr=0;
          }
     }
}
else
```

```
    {
        theSol.fp=0;
        theSol.sol=KBunit::no_record;
        refVecOfSols.push_back(theSol);
    }
    //now SimilarityVec contains  the iterator values which point to
        KB-Units which are similar based on PERCENTSIMILARITY, just
        sort it and draw out the KB-Units and return them
    std::sort(SimilarityVec.begin(), SimilarityVec.end(),
       SimilarityCount::greater());
    for(itSim=SimilarityVec.begin();itSim!=SimilarityVec.end();++
       itSim){
        theSol.fp=itSim->m_It->second.fp;
        theSol.sol=itSim->m_It->second.sol;
        refVecOfSols.push_back(theSol);
    }
    //Now moving the kbunits with -ve frequencies in the last
    vector<KBunit> temp;
    vector<KBunit>::iterator it;
    for(int i=0;i<refVecOfSols.size();++i){
        if(refVecOfSols[i].fp<0){
            temp.push_back(refVecOfSols[i]);
            it=refVecOfSols.begin();
            it=it+i;
            refVecOfSols.erase(it);
            i--;
        }
    }
    //now push the -ve frequency kbunits back in refVecOfSols
    for(int i=0;i<temp.size();i++)
        refVecOfSols.push_back(temp[i]);
    //Now remove all duplicates from the bottom of refVec
    //keep all highly matched and highly frequent, remove all others
    std::set<KBunit> s;
    std::vector<KBunit> vunique;
    std::pair<std::set<KBunit>::iterator,bool> ret;
    unsigned size = refVecOfSols.size();
    for( unsigned i = 0; i < size; ++i ){
        ret=s.insert(refVecOfSols[i]);
        if(ret.second)
            vunique.push_back(refVecOfSols[i]);
    }
    refVecOfSols.assign( vunique.begin(), vunique.end());
}
```

```
//Implementation of Algorithm 2
void Agent::SelectKBUnitOnHumanFactorBasis(double CurrentTime)
{
     if(m_bLockEvaluateScenarioMethod)
          return;
     int randomIndex=-1;
     int leave=0;
     int numSols;
     if(m_HFactor.ReflexAction())
     {
          std::list<Agent*>::const_iterator itOtherAgents=m_pWorld->
              Agents().begin();
          if(m_bSendAlarmMessageToOtherAgents)
          {
               int recieverid;
               for( ;itOtherAgents!=m_pWorld->Agents().end();
                    ++itOtherAgents)
               {
                    //if receiver and sender both are not same
                    if((*itOtherAgents)->ID()!=ID())
                    {
                         recieverid=(*itOtherAgents)->ID();
                         Dispatcher->
                              DispatchMsg(SEND_MSG_IMMEDIATELY,ID(),
                                   recieverid,Msg_ThereIsAHighHazard,
                                      NULL);
                    }
               }
          }
          m_bSendAlarmMessageToOtherAgents=false;
          return;
     }
     KBunit Solfound;
     Vector2D posfound;
     for(unordered_map<Vector2D, vector<KBunit>>::
          iterator itHash=m_pWM->m_HashTable.begin();
          itHash!=m_pWM->m_HashTable.end();++itHash)
     {
          numSols=(int)m_pWM->m_HashTable[itHash->first].size();
          GenProbabilities(numSols,m_HFactor.getProbability());
          fresults<<std::endl;
          for(unsigned int kk=0;kk<m_vProbabilities.size();++kk)
               fresults<<m_vProbabilities[kk]<<std::endl;
          boost::random::discrete_distribution<>
               binomialdist(m_vProbabilities);
          randomIndex=binomialdist(BiasedRandomNumGenerator);
          double s=MeasureSurprise(randomIndex);
          m_dSurprise=ScaleSurprise(s);
          MeasureIgnorance();
          m_dIgnorance=ScaleIgnorance(m_dIgnorance);
```

```
            for(leave=0;leave<numSols;++leave)
            {
                  if(leave==randomIndex)
                  {
                        posfound=itHash->first;
                        Solfound=itHash->second[leave];
                        m_pWM->m_HashTable[itHash->first].erase(
                              (itHash->second.begin())+leave);
                        m_pWM->m_HashTable[itHash->first].insert(
                              (itHash->second.begin()), Solfound);
                        m_bLockEvaluateScenarioMethod=true;
                        fresults<<endl;
                        for(unsigned int i=0;i<m_pWM->
                              m_HashTable[itHash->first].size();++i)
                        {
                              fresults<<m_pWM->
                                    m_HashTable[itHash->
                                    first][i].ToString()<<std::endl;
                        }
                        break;
                  }
            }
      }
}
```

Appendix E

The knowledgebase entries with associated CNDs

Table E.1: Initial or default entries of knowledge-base[2]with associated CNDs

Rule	CND	Description	KB-Unit	
			AP	f
R1	00000000010001001	F, LH, O, N	UW	0
R2	00000000010001001	F, LH, O, N	UF	0
R3	00000000010001001	F, LH, O, N	UD	-1
R4	00000000010001001	F, LH, O, N	UP	-1
R5	00000000010001001	F, LH, O, N	UDP	-1
R6	00000000010001001	F, LH, O, N	UWC	-1
R7	00000000010001001	F, LH, O, N	UM	-1
R8	00000000010001001	F, LH, O, N	UC	-1
R9	00000000110011001	F, HH, O, E, N	GM	0
R10	00000001110011001	F, HH, O, E, Fu, N	GM	0

Continued on next page

[2]'F' stands for fire, 'LH' for low hazard, 'HH' for high hazard, 'O' means ordinary combustibles, 'Fu' represents fuel lines, 'E' stands for power lines, 'M' is to mean combustable metals, 'A', 'B', 'C', & 'D' are fire classes, 'PAl' is the process alarm, 'MAl' is the muster alarm and 'N' means 'no information about fire class'.

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R11	00000010010011001	F, HH, O, M, N	GM	0
R12	00000011000011001	F, HH, Fu, M, N	GM	0
R13	00000011010011001	F, HH, O, Fu, M, N	GM	0
R14	00000011110011001	F, HH, O, E, Fu, M, N	GM	0
R15	00001000000011001	F, HH, A	GM	0
R16	00001000010001001	F, LH, O, A	UW	0
R17	00001000010001001	F, LH, O, A	UF	0
R18	00001000010001001	F, LH, O, A	UC	-1
R19	00001000010001001	F, LH, O, A	UD	-1
R20	00001000010001001	F, LH, O, A	UP	-1
R21	00001000010001001	F, LH, O, A	UM	0
R22	00001000010001001	F, LH, O, A	UDP	-1
R23	00001000010001001	F, LH, O, A	UWC	-1
R24	00001000110001001	F, LH, O, E, A	UM	0
R25	00001000110011001	F, HH, O, E, A	GM	0
R26	00001001010001001	F, LH, O, Fu, A	UF	0
R27	00001001010001001	F, LH, O, Fu, A	UM	0
R28	00001001110001001	F, LH, O, E, Fu, A	UM	0
R29	00001001110011001	F, HH, O, E, Fu, A	GM	0
R30	00001010010011001	F, HH, O, M, A	GM	0
R31	00001011000011001	F, HH, Fu, M, A	GM	0

Continued on next page

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R32	00001011010011001	F, HH, O, Fu, M, A	GM	0
R33	00001011110011001	F, HH, O, E, Fu, M, A	GM	0
R34	00010000000011001	F, HH, B	GM	0
R35	00010000110011001	F, HH, O, E, B	GM	0
R36	00010001000001001	F, LH, Fu, B	UW	-1
R37	00010001000001001	F, LH, Fu, B	UF	0
R38	00010001000001001	F, LH, Fu, B	UC	0
R39	00010001000001001	F, LH, Fu, B	UD	0
R40	00010001000001001	F, LH, Fu, B	UP	0
R41	00010001000001001	F, LH, Fu, B	UM	0
R42	00010001000001001	F, LH, Fu, B	UDP	-1
R43	00010001000001001	F, LH, Fu, B	UWC	-1
R44	00010001010001001	F, LH, O, Fu, B	UF	0
R45	00010001010001001	F, LH, O, Fu, B	UM	0
R46	00010001100001001	F, LH, E, Fu, B	UC	0
R47	00010001100001001	F, LH, E, Fu, B	UD	0
R48	00010001100001001	F, LH, E, Fu, B	UP	0
R49	00010001100001001	F, LH, E, Fu, B	UM	0
R50	00010001110001001	F, LH, O, E, Fu, B	UM	0
R51	00010001110001001	F, LH, O, E, Fu, B	UM	0
R52	00010001110011001	F, HH, O, E, Fu, B	GM	0

Continued on next page

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R53	00010010010011001	F, HH, O, M, B	GM	0
R54	00010011000011001	F, HH, Fu, M, B	GM	0
R55	00010011010011001	F, HH, O, Fu, M, B	GM	0
R56	00010011110011001	F, HH, O, E, Fu, M, B	GM	0
R57	00011000000011001	F, HH, C	GM	0
R58	00011000100001001	F, LH, E, C	UW	-1
R59	00011000100001001	F, LH, E, C	UF	-1
R60	00011000100001001	F, LH, E, C	UC	0
R61	00011000100001001	F, LH, E, C	UD	0
R62	00011000100001001	F, LH, E, C	UP	0
R63	00011000100001001	F, LH, E, C	UM	0
R64	00011000100001001	F, LH, E, C	UDP	-1
R65	00011000100001001	F, LH, E, C	UWC	-1
R66	00011000110001001	F, LH, O, E, C	UM	0
R67	00011000110011001	F, HH, O, E, C	GM	0
R68	00011001100001001	F, LH, E, Fu, C	UC	0
R69	00011001100001001	F, LH, E, Fu, C	UD	0
R70	00011001100001001	F, LH, E, Fu, C	UP	0
R71	00011001100001001	F, LH, E, Fu, C	UM	0
R72	00011001110011001	F, HH, O, E, Fu, C	GM	0
R73	00011010010011001	F, HH, O, M, C	GM	0

Continued on next page

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R74	00011011000011001	F, HH, Fu, M, C	GM	0
R75	00011011010011001	F, HH, O, Fu, M, C	GM	0
R76	00011011110011001	F, HH, O, E, Fu, M, C	GM	0
R77	00100000000001001	F, LH, D	UDP	0
R78	00100000000011001	F, HH, D	GM	0
R79	00100000110011001	F, HH, O, E, D	GM	0
R80	00100001010001001	F, LH, O, Fu, D	UDP	0
R81	00100001110001001	F, LH, O, E, Fu, D	UDP	0
R82	00100001110011001	F, HH, O, E, Fu, D	GM	0
R83	00100010010001001	F, LH, O, M, D	UDP	0
R84	00100010010011001	F, HH, O, M, D	GM	0
R85	00100011000011001	F, HH, Fu, M, D	GM	0
R86	00100011010011001	F, HH, O, Fu, M, D	GM	0
R87	00100011110011001	F, HH, O, E, Fu, M, D	GM	0
R88	00000000000000010	PAl, N	PA	0
R89	01000000010001001	F, LH, O, PAl, N	UW	0
R90	01000000010001001	F, LH, O, PAl, N	UC	-1
R91	01000000010001001	F, LH, O, PAl, N	UF	0
R92	01000000010001001	F, LH, O, PAl, N	UD	-1
R93	01000000010001001	F, LH, O, PAl, N	UP	-1
R94	01000000010001001	F, LH, O, PAl, N	UDP	-1

Continued on next page

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R95	01000000010001001	F, LH, O, PAl, N	UWC	-1
R96	01000000010001001	F, LH, O, PAl, N	UM	-1
R97	01000000010011001	F, HH, O, PAl, N	PA	0
R98	01000000100011001	F, HH, E, PAl, N	PA	0
R99	01000001000011001	F, HH, Fu, PAl, N	PA	0
R100	01000010000011001	F, HH, M, PAl, N	PA	0
R101	01001000000001001	F, LH, PAl, A	CP	0
R102	01001000010001001	F, LH, O, PAl, A	CP	0
R103	01001000010011001	F, HH, O, PAl, A	PA	0
R104	01001000100001001	F, LH, E, PAl, A	CP	0
R105	01001000100011001	F, HH, E, PAl, A	PA	0
R106	01001000110001001	F, LH, O, E, PAl, A	CP	0
R107	01001001000001001	F, LH, Fu, PAl, A	CP	0
R108	01001001000011001	F, HH, Fu, PAl, A	PA	0
R109	01001001010001001	F, LH, O, Fu, PAl, A	CP	0
R110	01001001010001001	F, LH, O, Fu, PAl, A	CP	0
R111	01001001110001001	F, LH, O, E, Fu, PAl, A	CP	0
R112	01001010000001001	F, LH, M, PAl, A	CP	0
R113	01001010000011001	F, HH, M, PAl, A	PA	0
R114	01010000010011001	F, HH, O, PAl, B	PA	0
R115	01010000100011001	F, HH, E, PAl, B	PA	0

Continued on next page

Table E.1 – continued from previous page

Rule	CND	Description	KB-Unit	
			AP	f
R116	01010001000011001	F, HH, Fu, PAl, B	PA	0
R117	01010010000011001	F, HH, M, PAl, B	PA	0
R118	10001000000001001	F, LH, MAl, A	GM	0
R119	10001000010001001	F, LH, O, MAl, A	GM	0
R120	10001000100001001	F, LH, E, MAl, A	GM	0
R121	10001001000001001	F, LH, Fu, MAl, A	GM	0
R122	10001010000001001	F, LH, M, MAl, A	GM	0
R123	11000000010011001	F, HH, O, N	GM	0
R124	11000000100011001	F, HH, E, N	GM	0
R125	11000001000011001	F, HH, Fu, N	GM	0
R126	11001000010001001	F, LH, O, A	GM	0
R127	11001000010011001	F, HH, O, A	GM	0
R128	11001000100001001	F, LH, E, A	GM	0
R129	11001000100011001	F, HH, E, A	GM	0
R130	11001001000001001	F, LH, Fu, A	GM	0
R131	11001001000011001	F, HH, Fu, A	GM	0
R132	11001010000001001	F, LH, M, A	GM	0
R133	11001010000011001	F, HH, M, A	GM	0
R134	11010000010011001	F, HH, O, B	GM	0
R135	11010000100011001	F, HH, E, B	GM	0
R136	11010001000011001	F, HH, Fu, B	GM	0