Cognitively Inspired Decision Making for Software Agents: Integrated Mechanisms for Action Selection, Expectation, Automatization and Non-Routine Problem Solving

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Memphis

Aregahegn Seifu Negatu

August 2006

To the Graduate Council:

I am submitting herewith a dissertation written by Aregahegn Seifu Negatu entitled "Cognitively Inspired Decision Making for Software Agents: Integrated Mechanisms for Action Selection, Expectation, Automatization and Non-Routine Problem Solving." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy with a major in Computer Science.

Stan Franklin

Stanley P. Franklin, Ph.D.
Major Professor

We have read this dissertation and recommend its acceptance:

[illegible]

Arthur C. Graesser, Ph.D.

[illegible]

Thomas L. McCauley, Ph.D.

[illegible]

Sajjan G. Shiva, Ph.D.

[illegible]

Junmei Zhu, Ph.D.

Accepted for the Council:

Karen Weddle-West

Karen D. Weddle-West, Ph.D.
Assistant Vice Provost for Graduate Studies

DEDICATION

To Meaza Sinekirstos

For all of her struggles

Acknowledgements

The work with this dissertation has been extensive and demanding, but primarily exciting, instructive, and amusing. Without help, support, and encouragement from several persons, I would never have been able to conclude this work.

First and foremost I would like to express my deep gratitude to my supervisor Dr. Stan Franklin, for his inspiration, guidance, patience, encouragement and commitment to my research, for his friendship, and for establishing and leading the Cognitive Computing Research Group (CCRG) at the University of Memphis. He has given me good advice and direction while allowing me to explore in my own way.

My thanks also go to the members of my dissertation committee, Dr. Arthur C. Graesser, Dr. Thomas Lee McCauley, Dr. Sajjan G. Shiva, and Dr. Junmei Zhu for providing invaluable input and comments that improved the presentation and contents of this dissertation. Particularly, I would like to thank Dr. Graesser for sharing with me many concepts of cognitive modeling; I would also like to thank Dr. McCauley for his collaboration and coauthoring that resulted from many weekly meetings he has had with me and Dr. Franklin.

I would like to thank members of Cognitive Computing Research Group for their friendship, for many inspiring group and one-to-one discussions related to my research. I am particularly indebted to Uma Ramamurthy for being a wonderful friend. I have also benefited from discussions I have with our research group collaborator, Dr. Barney Baars.

I would like to express my gratefulness to all my, to many to list, wonderful formal teachers through the years: Thanks to all at St. Paul Junior High, Kolfe Comprehensive High, and Addis Ababa Technical schools for laying down my academic foundation and sparking my curiosity in science and engineering. Thanks also to all at Engineering Faculty of Addis Ababa University, Indian Institute of Science, and University of Memphis for giving me the knowledge and love for electrical engineering, computer engineering and computer science.

My deepest gratitude goes to my best informal teacher – my mother. She shaped me to who I am now. She taught me hard working by example. I am eternally indebted to her for her sacrifices to support my early education and my morally nourished upbringing.

Last, but not least, I would like to thank my wife Frehiwot for her understanding and love during the past few years of my research, during which we also have been blessed with our two sons – Surafiel and Yonas. I am grateful for their unconditional love and for showing me the wonderful experience of a warm and calming family life.

Abstract

Negatu, Aregahegn Seifu. Ph.D. The University of Memphis. August, 1996. Cognitively Inspired Decision Making for Software Agents: Integrated Mechanisms for Action Selection, Expectation, Automatization and Non-Routine Problem Solving. Major Professor: Stanley P. Franklin, Ph.D.

Despite impressive advances in the past decades, autonomous agents living in dynamic and unpredictable environments are typically equipped with simple decision-making mechanisms in their sense-decide-act routines. These agents deal mostly with one goal at a time. This research aspires to model, design and/or implement a sophisticated decision making mechanism that selects the agent's next action with different levels of awareness: automatized skills, consciously mediated routine solutions, and consciously deliberated non-routine solutions. Such a decision-making mechanism is presented in a "conscious" software agent framework called IDA that implements Baars' Global Workspace Theory of consciousness. IDA integrates many computational and conceptual mechanisms, among which this research deals with its action selection, expectation, automatization and non-routine problem solving modules.

The overarching continual task of an agent's intelligence is for the service of choosing, at each moment in time, the appropriate action in response to exogenous and endogenous stimuli. IDA's action selection mechanism (ASM) can interleave and prioritize actions of different and competing goal hierarchies. The ASM system is implemented as a domain independent and reusable framework for behavior networks and is tested as a controller to a khepera robot operating in a real world domain.

We humans have the amazing ability to learn a procedural task (e.g. walking) so well that we do not need to think about the task consciously in order to accomplish it. This ability is what we call automatization. Once a task has been automatized there is no need for attention to be paid to its execution unless the expected result does not occur. At failure of expectation, deautomatization process temporarily disables the automatization effects and "conscious" control plays a role to deal with the failure situation. We implement the automatization and deautomatization cognitive functions as a self-organizing system in the IDA framework.

Non-routine problem solving is the ability to devise unexpected, and often clever, solutions to problems that never been encountered before. We will present a detailed design and specification of a non-routine problem solving mechanism as a special goal context hierarchy that guides a deliberative solution search process, which we will discuss in IDA's cognitive cycle.

TABLE OF CONTENTS

List of Tables

List of Figures

1 Introduction

Despite impressive advances in the past decade, autonomous agents living in dynamic and unpredictable environments are typically equipped with simple decision-making mechanisms in their sense-decide-act routines. These agents deal mostly with one goal at a time. This research aspires to model, design and/or implement a dynamic-decision making mechanism that selects the agent's next action with different levels of awareness: automatized skills, consciously mediated routine solutions, and consciously deliberated non-routine solutions. Such a decision-making mechanism is presented in a "conscious" agent framework.

This introductory chapter will cover background concepts that set the context in which to define our research goals. We outline our research objectives by raising relevant questions and by stating goals that will answer the questions. We conclude the chapter by outlining a road map to this dissertation.

1.1 Consciousness

In our wakeful states, we, humans, are conscious of some state of mind; i.e., we are aware of being situated in space and time; we are aware of our thoughts, feelings, and intentions; we remember the past and contemplate the future; and we may be aware of our actions and expectations. We are not conscious when we are in a deep sleep, in a coma, or under general anesthesia. For a long time, consciousness, and its scientific study, was ignored by mainstream psychology. Although there is no agreed upon definition for the term "consciousness," it has been the topic of scientific investigation over the last two decades.

Block (2002) tries to explain the different aspects of consciousness. He points out that consciousness is a hybrid concept with four distinct components: phenomenal consciousness, access consciousness, self consciousness, and monitoring consciousness.

Phenomenal consciousness is a qualitative experience whose properties are those of experiences, which include bodily sensations, perception, feelings, emotions, and thoughts. Phenomenal conscious states are "what is it like" states of having experiential properties such as when one sees, hears, smells, tastes, and has pain. For instance, what is it like to experience the color (red, yellow, etc.) of a rose, or the sweetness of sugar?

Access consciousness is a representation in the brain. The contents of access consciousness are broadcast for use in high-level processing such as reasoning, problem solving, and rational guidance of action (including reporting). Block uses the term "broadcast" in the same sense as Baars' (1998) global workspace theory: conscious representations are those whose contents are broadcast in a global workspace. Access consciousness is similar to the Dennett's (1993) notion of consciousness as fame (global access) in the brain. Block emphasizes that access consciousness captures the notion of functional consciousness (realizable as an information processing computational system) and that it is relevant to the notion of consciousness as used in cognitive neuroscience.

Self consciousness refers to awareness of oneself. It is having the concept of self and using this concept in thinking about oneself. Chimps seem to recognize that they see themselves in mirrors (Povinelli, 1994). Human babies show this behavior only after their eighteenth month. This may be a test for self-consciousness.

Monitoring consciousness refers to the awareness of percepts as distinct from the percepts themselves. This is similar to a metacognitive notion, in the sense of having a thought (inner perception) about one's conscious state and differing from the conscious state itself.

Considering phenomenal consciousness and access consciousness, can one exist without the other? According to Block, and we agree, quite often they occur together. Further, at least conceptually, it is possible to have one without the other, although it is unclear whether dissociations of phenomenal consciousness and access consciousness actually happen.

Another issue to consider is the possibility of scientifically accounting for phenomenal consciousness and the ability to build a computational model to realize it. This is the so called "hard problem" (Chalmers, 1996), or the problem of the "explanatory gap" (Levine, 1993). Different computational models have been proposed and attempted to explain the various concepts of consciousness (Sun & Franklin, 2004) and eventually implement a "conscious" machine (Franklin, 2003).

Dennett does not believe in the existence of phenomenality (qualia), but instead states that the functional view of consciousness explains the phenomenal view of consciousness. There is a wider agreement that consciousness involves cognitive processes and the first-person (subjective) phenomena (also called sentience.) Conscious contents are exceedingly numerous; including perceptions (many modalities), attention, learning, problem solving, emotions, motivations, intention (goal images), and many other cognitive processes. Presence or absence of conscious access or conscious experience is associated with each cognitive process and its associated attributes and usage.

There are properties that are common to the various conscious contents. According to Baars (1998), a defining property or behavioral measurement of phenomenal consciousness in humans is accurate reporting of events in one's awareness. Reporting can happen verbally or with any other voluntary response. Baars also suggests additional criteria for consciousness that include global distribution of conscious contents, internal consistency, informativeness of conscious content, possibly interaction with self-system or subjectivity of conscious experience, limited-capacity and seriality, facilitation for learning, awareness to knowledge for voluntary action selection, exhibiting perceptual bias, and so on. These criteria are not independent of each other. These and other valid criteria for consciousness could be useful in evaluating how "conscious" artifacts like IDA (section 1.4.3) fare in modeling human consciousness.

1.2 Global Workspace Theory

The global workspace (GW) theory, a psychological theory of consciousness, is firmly rooted in empirical cognitive science and in neuroscience (Baars, 1988, 1997, 2002). Bernard. J. Baars, one of the primary investigators in the scientific study of consciousness, developed this theory by treating consciousness as a variable and comparing different phenomena that show its presence and absence. Contrastive analyses, which provide important empirical bases, are used by many investigators (most are cognitive scientists and cognitive neuroscientists) to study conscious and unconscious phenomena, even as some avoid the term "consciousness" in their discussion of the phenomena. Alias terms such as "explicit versus implicit cognition," "strategic versus automatic control," and "novel versus routine events," have been employed. Baars (2000) identifies many more, which are partially reproduced in Table 1-1.

1.2.1 Contrastive Properties: Conscious versus Unconscious

Based on the contrastive analysis, Baars identifies that conscious aspects are associated with a limited capacity, while unconscious aspects are associated with relative vastness. Immediate memory, selectivity of attention and strategic control are examples of limited capacity mechanisms. Such limited capacity mechanisms in the brain seems to be slow/inefficient, serial, internally consistent, and error-prone.

The brain is a massive collection of networked processes, each specializing in a specific task. These processes are mostly unconscious and very efficient in executing routine tasks without error. They operate in parallel and vast collective capacity. Baars [1998] states that the central puzzle is to find out how a serial, coherent, and limited conscious content (neuronal activity) emerges from mostly unconscious, parallel, massively differentiated and relatively unlimited society of brain processes. He explains the puzzle as a requirement of *global accessibility* to conscious contents. That is, consciousness is a publicity organ of the brain or fame in the brain (Dennett, 1993) that facilitates the central dissemination of information towards global coordination and control.

Table 1-1: Some widely studied polarities between conscious and unconscious phenomena (partially reproduced from Baars, 2000).

	Conscious	Unconscious
1	Explicit cognition	Implicit cognition
2	Immediate memory	Long-term memory
3	Novel, informative, and significant events	Routine, predictable, and non-significant events
4	Attended information	Unattended information
5	Declarative memory (facts)	Procedural memory (skills)
6	Effortful tasks	Spontaneous/automatic tasks
7	Remembering (recall)	Knowing (recognition)
8	Strategic control	Automatic control
9	Grammatical strings	Implicit underlying grammars
10	Rehearsed items in working memory	Unrehearsed items
11	Wakefulness and dreams (cortical arousal)	Deep sleep, coma, sedation (cortical slow waves)
12	Explicit inferences	Automatic inferences
13	Episodic memory (autobiographical)	Semantic memory (conceptual knowledge)
14	Intentional learning	Incidental learning

1.2.2 Basic Components of Global Workspace Theory

Global Workspace (GW) theory models the mobilization and integrative function of consciousness. Consciousness creates a global access, helping to recruit and integrate the many separate and independent brain functions or unconscious collection of knowledge resources. They are recruited internally, but partially driven by stimulus input. GW theory (see Figure 1-1) has three basic constructs. We will briefly discuss each of them below.

1.2.2.1 Global Workspace

Global workspace is the central construct of this theory. In AI terms, global workspace is a globally accessible block of working memory that mediates information exchange and novel interaction among individual processors in a distributed-processing system. GW theory proposes the same structure to support conscious experience via global accessibility. The global workspace is accessible to most specialized processors and also broadcasts (or disseminates) its contents in such a way that every processor receives the contents. The broadcast is of the conscious content. Multiple specialized processors may compete for access to the global workspace. Others may cooperate (or form coalitions) to broadcast their contents as a global message. The global workspace has a limited-capacity and serial processing, and, as such, content of a single coalition of processors can be broadcast globally at one time. The candidates for conscious content have a wide range since the content of most unconscious specialized processors can compete for "consciousness".

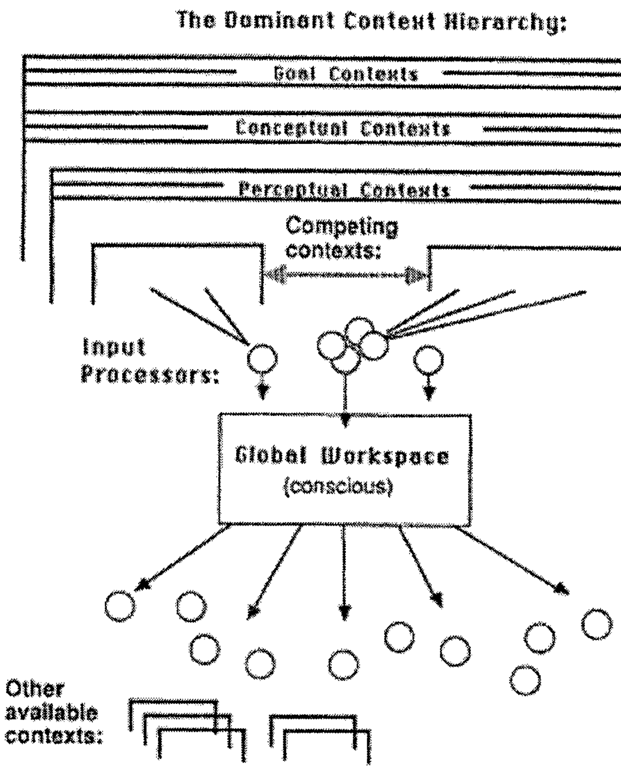

Figure 1-1: Global workspace theory. The depiction shows dominant context hierarchy, and competition and cooperation among different contexts.

1.2.2.2 Special Processors

GW theory postulates that cognition is implemented by a multitude of relatively small, special purpose processors or networks, almost always unconscious. It is a multi-agent system, similar to Edelman's repertoire of neuronal groups (1987), Minsky's agents (1985), Ornstein's small minds (1986), and Jackson's demons (1987). Each specialized processor is autonomous and has focused expertise towards either detecting a feature or performing a primitive action. Processors in GW theory correspond to neuronal groups or cell assemblies. But, autonomous and distributed processing is common place in many levels including at cell level. Each cell, depending on its physiological location, reads its DNA instruction to recognize its special role.

Specialized processors cooperate or form coalitions and perform a vast number of unconscious or automatized tasks efficiently in parallel. Respiratory and blood circulatory systems involve many unconscious, specialized coalitions of processors, which operate mostly in parallel with high efficiency and with few or no errors.

Processors have a potential to bring their content to "consciousness." They compete based on their activation level, which in turn depends on their relevance to the current state of an agent and of their content. Coalitions with novel contents have stronger activation levels than those with routine content (less information value). The chance for conscious access grows with activation levels. High activation is necessary, but may not be sufficient to access "consciousness."

1.2.2.3 Contexts

Contexts are relatively stable (over time) coalitions of unconscious processors that constrain or shape "consciousness." That is, they evoke and shape global messages or conscious contents without themselves becoming conscious. There are different types of contexts including goal contexts, perceptual contexts, conceptual contexts, and cultural contexts. Goal contexts constrain conscious goal images or intentions – mental representations of one's own future actions. Perceptual contexts constrain conscious experiences related to perception. Conceptual contexts constrain conscious access to abstract concepts. Cultural contexts constrain conscious access to social interactions.

Contexts can compete or cooperate to jointly constrain the next conscious contents. Contexts are also hierarchical and, as such, a context is nested under other contexts. Contexts in the same level of hierarchy compete with each other. Nested contexts cooperate with each other. The effect of the inner context in the hierarchy assists the constraining effect of the higher level contexts. At any given instant, one context hierarchy is dominant (controls current access to the global workspace). Based on the nesting structure and dominance of a context, Baars defines context hierarchy, dominant context hierarchy, dominant goal context, and dominant goal context hierarchy.

A goal hierarchy may have a multilevel goal structure that consists of goals and subgoals. Goals at each level of the hierarchy can be considered goal contexts. To satisfy the higher-level goals, their subgoals must be achieved.

1.2.3 A Working Theatre

Baars (1997) uses the 'theatre metaphor' to explain the working of global workspace theory. In a working theatre of "consciousness," one can identify the following: a stage, a spotlight, actors or processors that compete for spotlight, stage managers who are behind the scene influencing what comes under the spotlight, and an audience of specialized processors.

A stage setting contains various information pieces, but only the events under the bright spotlight are entirely conscious. More activity takes place in the production and stage setting than what happens under the spotlight. Conscious events are shaped by behind the scene supporters, context setters or unconscious processors. The spotlight selects the most significant actors on the stage. Actors under the spotlight perform their acts, and the audio/visual message is broadcast to an audience – the unconscious processors. Each audience member receives the broadcast and interprets the message in its own way. The broadcast also reaches behind the scene operators and prepares them to influence what is presented on stage and under the spotlight in the next act. Many acts happen unconsciously in the audience, in the backstage, and in the dark part of the stage.

In GW theory, conscious content is the content of a significant coalition of processors. Significance is related to the information value associated with novel situations. The limited-capacity and seriality is enforced by the fact that the spotlight shines only on a single coalition of processors at a time. Each unconscious processor (audience member or stage manager), when it finds the global message to be relevant (a local decision), performs its specific function.

1.2.4 Conclusion

Global Workspace (GW) theory (Baars, 1988, 1997, 2002), based on its three constructs, explains many cognitive functions including attention, action selection, expectation, automatization, learning, problem solving, emotion, voluntary action, metacognition, and a sense of self. GW theory presents an integrated model of high level cognition based on the premise that the brain is a collection of unconscious specialized processors, and that global access to coherent and dominant information is a necessary condition for conscious access.

Baars (1988) developed the theory nearly two decades ago using extensive but indirect evidence that was available at the time. Since then, and as reported by Baars (2002), there has been a steady accumulation of evidence and growing consensus by many researchers (Edelman & Tononi, 2000; Dehaene & Naccache, 2001; Dennett, 2001; Kanwisher, 2001; Rees, 2001; and others) to support the global accessibility of conscious states. The research discussed in this dissertation is part of the IDA (Information Distribution Agent) project that implements a software agent system using the framework of GW theory.

1.3 "Conscious" Machines

In recent years, there has been a revival of the scientific study of "consciousness." Particularly, advances in cognitive psychology, cognitive neuroscience and computer technology have inspired many computational models of "consciousness." Besides Baars' (1988, 1997, 2002) Global Workspace theory, many other theories of consciousness have been proposed, for instance, by D.L. Schacter (1989), Daniel Dennett (1991), P. Caruthers (1996), Igor Alexander (1996), E.T. Rolls (1998), John G. Taylor (1998), M. O'Brien and J. Opie (1999), Antonio Damasio (2000), Gerald M. Edelman and Giulio Tononi (2000), and Geral Sommerhof (2000). These theories attempt to explain some aspects of consciousness (phenomenal, access, monitoring, and self) and all contribute to our understanding of consciousness. These theories suggest the possibility for building machine "consciousness".

There are arguments against machine "consciousness," mainly from agnostics (primarily based on the dualistic view of the mind-body problem) about the possibility of a computational accounting of phenomenal consciousness - the "hard problem." On the other side, as the different computational models try to explain, the brain is a biological machine, and consciousness is a process operating on mental representations or is an intrinsic property of mental representations inside that machine; so consciousness as such is available for scientific study and computational modeling. As the next logical step of the computational modeling, there have been some endeavors to implement "conscious" machines. One example is Igor Alexander's (2000) MAGNUS system, a neural state machine in which "consciousness" arises from iconic neural firing patterns. The firing patterns are meaningful in relation to sensory input. Another example is our own IDA model, which we will give a brief introduction to below and have detailed discussions of its conceptual and computational models in the coming chapters.

1.4 IDA: A "Conscious" Software Agent

1.4.1 Autonomous Agents

What is an autonomous agent? The study of autonomous agents is the latest endeavor to model and develop a system that exhibits multiple characteristics that are associated with intelligence behavior such as that in humans. Early artificial intelligence (AI) researchers, enthused by expectations of the early computer age and their early results, set out to construct complete intelligent systems. Such AI systems were expected to sense and perceive their environment, to reason and solve problems, to act and interact to achieve their agenda, to learn from experience. But coming up with a complete system was found to be difficult, even in a toy environment. As a result, AI research shifted to the individual cognitive functions of intelligent systems and their applications in real world problems. These functions include perception, natural language understanding, learning, problem solving, planning and action selection. Usually AI researches have been coupled with established results in cognitive science, cognitive neuroscience, linguistics, statistics, dynamical systems, ethology, and other fields of study. Advances in each of these areas have enabled contemporary computer science researchers to build artifacts that integrate capabilities associated with multiple intelligent functions. Such artifacts are called autonomous (intelligent) agents.

For the most part, depending on the type of agent (based on domain and incorporated cognitive models) they built, many researchers advanced their own definitions for autonomous agents (Brustoloni, 1991; Smith et. al, 1994; Hays-Roth, 1995; Russel & Norving 1995; Wooldrige & Jennings, 1995; Franklin & Grasser, 1996). As defined by Franklin and Graesser (1996), "An *autonomous agent* is a system situated within, and as part of, an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future." This definition is relatively succinct in capturing the essence of agents and it is getting a wider acceptance in the research community.

1.4.2 Software Agents

Software agents are types of non-biological autonomous agents that live in a computational environment as software entities. The computational environment may include an operating system, a network (and many associated protocols such as the World-Wide Web), database systems, and many other computing and device control systems. Such computational environments present a complex and dynamic real world problem for a software agent to deal with. Many are developed to assist humans in various tasks such as computer system administration (e.g.: Song, Franklin, & Negatu (1996)), mining and retrieval of relevant information in the world-wide web (many web-crawlers) and in database systems, easing the use of computer interfaces (example WS Windows helper agent), making the lives of computer users difficult (many computer viruses), and others.

1.4.3 "Conscious" Software Agents

The addition of "consciousness" mechanism in a software agent is expected to lead to a more robust, more human-like decision making and more creative problem solving agent system. We define a "conscious" software agent as an autonomous agent (Franklin & Graesser, 1997) that implements Baars' global workspace theory. IDA (Intelligent Distribution Agent) is a "conscious" software agent that was developed for the U.S. Navy (Franklin, 2001; Franklin, Kelemen, & McCauley, 1998). The general principle in our agent design is: if you want smart software, copy it from humans. As of this writing, IDA has been successfully demonstrated to the Navy. IDA's technology is being used to develop a product. IDA is "conscious" in the sense that it has functional or access "consciousness" (Franklin, 2003) with no claim of sentience or phenomenal consciousness.

1.4.4 IDA's Domain

At the end of each sailor's tour of duty, he or she is assigned to a new billet. This assignment process is called distribution, hence the name. The Navy employs some 280 people, called detailers, to effect these new assignments. IDA's task is to facilitate this process by completely automating the role of detailer. IDA must communicate with sailors via email in natural language, understanding the content and producing life-like responses. Sometimes IDA will initiate conversations and must access several databases, again understanding the content. IDA must see that the Navy's needs are satisfied by adhering to a number of Navy policies and must hold down moving costs. IDA must see that the requirements for each job are met, as well as cater to the needs and desires of the sailor as much as is possible. This includes negotiating with the sailor via email in natural language. Finally, she must make the decision of a new job for the sailor.

1.5 Research Objectives

1.5.1 Goals of the IDA Project

The Cognitive Computing Research Group (CCRG) has been building a "conscious" software agent called IDA (see section 1.4). As we will see in chapter 2, IDA aspires to model human-like minds by integrating different mechanisms. Cognitive science researchers have a mission to explain how the mind works and often propose conceptual and computational models for the different cognitive functions. Computer science researchers tend to model and build agents with human-like intelligent behavior, but with mechanisms that may not correspond to those occurring in humans. So although IDA is a cognitively inspired system, it will have gaps in its cognitive modeling in two ways:

- First, there is a gap between computer science mechanisms in agent systems and the corresponding cognitive functions modeled by each mechanism. In IDA we try to narrow this gap by continuously modifying existing mechanisms to make them more plausible in representing cognitive processes.

- Second, human intelligence encompasses a large number of cognitive processes, and an agent system often starts by integrating a few of the cognitive functions and fill the gaps incrementally.

The design and implementation approach in IDA is a recursive process that incorporates engineering and scientific methodologies. The approach allows us to explore the design and niche spaces and their interaction in the sense of Sloman (1993, 1995). The engineering methodology deals with: (a) a requirements specification (information-level description) with the capability of the agent such as the need for deliberation in the IDA domain, (b) a design specification to conceptually accommodate or integrate requirements in the agent architecture such as adding a conceptual mechanism for procedural learning in IDA, and (c) an implementation or a detailed implementation specification to incorporate and realize the capabilities.

Scientific methodology deals with: (a) a qualitative and quantitative analysis of the design and implementation, and an evaluation of how well the implementation meets the requirements; the analysis may include the identification of important hypotheses that comes out of the implementation, and (b) analysis and consideration of alternative designs in a design-space, which allows scientific validation and better understanding of alternative approaches that may improve design and implementations specifications. Scientific methodology can generate new requirements specifications that can be used to narrow the gaps discussed above.

The Cognitive Computing Research Group (this writer has been a member since its inception) has the objective of implementing an agent technology with a real-world application. Although IDA was developed as a distribution agent for the Navy, its technology can be applied to different information processing domains such as travel agencies and customer call centers (Franklin, 2001). Each implementation decision presents a hypothesis on how mind works. The research group has the objective and the hope of making scientific contributions by providing testable hypotheses that could be useful in cognitive science and cognitive neuroscience.

1.5.2 Objectives of the Research

This writer's research objectives deal with the action selection mechanism, the automatization and expectation mechanisms, and the non-routine problem solving mechanism of IDA; each of these mechanisms will be briefly introduced. These objectives raise a number of research questions that will be answered in this dissertation.

1.5.2.1 Action Selection Mechanism

We humans interact with our environment in ways to satisfy our agenda. In doing so, we do perceive, learn, remember, solve problems, plan, etc. All these capabilities are useless unless they produce actions. In general, the mind has many functions. But its overriding task is to generate and control appropriate behavior, or to produce the next action (Franklin, 1995). Albus (1981) supported this fact by stating that brains, even those of the tiniest insects, generate and control behavior.

In general, any agent should solve the action selection problem - the problem of selecting the appropriate action so as to satisfy its primary drives. The agent's intelligence is for the service of choosing, at each moment in time, the appropriate action with regard to all types of exogenous and endogenous stimuli. The action selection mechanism (ASM) of an agent specifies the methods of choosing appropriate actions.

In a rough decomposition of functions of the mind as a control structure, autonomous agents continuously perceive, select action, and execute the selected action. An action selection mechanism is part of the control structure with different control states, which include goals, behaviors, and plans. Goals are representations of future states of affairs that are motivationally directed and help to recruit and guide subgoals and motor systems to reach that state (Baars, 1988). Behaviors are modules with action patterns that respond to internal motivational state or external stimuli (reflexive actions). Plans are structures that guide a sequence of control states such as goals and behaviors towards a goal state. Such control states are usually incorporated in goal structures.

In the IDA model, the action selection mechanism is a goal structure system implemented based on Maes' behavior network (1989), which is covered in chapter 3. Goals and behaviors in an instantiated behavior network are goal contexts that constrain goal-images without themselves being conscious. Goal structures, also called behavior streams, contain goals and behaviors. They realize hierarchical goal contexts of the GW theory. In chapter 4, we will describe the details of IDA's action selection mechanism and related behavior-based action selection models.

In the design and implementation of IDA's action selection mechanism, the following questions are raised.

- What is the appropriate representation for the behavior network?
- How do behaviors control the internal and external motor actions?
- What is the mechanism that recruits goal hierarchies relevant to conscious broadcasts?
- How do behaviors and goals, as goal contexts, influence access to "consciousness?"
- How can the mechanism accommodate unconscious, voluntary and consciously mediated action types?
- What is a possible mechanism that can control priorities among active goal context hierarchies?
- What are the domain independent architectural features of a behavior network and can we build a mechanism that could lend itself to be a reusable action selection framework?
- How can we simplify the development of an action selection system?

Here we will see how "consciousness" helps to recruit and control actions. The action selection mechanism provides the base system so that conscious goals can activate unconscious routines in the goal contexts system to carry out voluntary actions, a sequence of consciously mediated actions or a sequence of automatic tasks. We will demonstrate the IDA's "consciously" mediated decision making process in a warehouse robot domain. Particularly, we will make GOMS (Goals, Operators, Methods, and Selection rules) analysis to specific domain task and demonstrate how the base action selection mechanism of IDA could be tuned to control priorities among competing goal context hierarchies.

1.5.2.2 Expectation and Automatization Mechanisms

A behavior, as a production rule, has preconditions and effects. In general, rules have multiple firing criteria including the satisfaction of their preconditions. A behavior fires when (a) its preconditions are satisfied, (b) it has the highest activation level, and (c) its activation level is over a threshold. The effects represent a desired or an expected outcome. In many implementations, it is assumed that after associated operations are completed, the expected effect is fulfilled. This assumption will not be valid if an agent operates in a dynamic environment. A real-world agent requires an independent feedback mechanism to monitor and report the actual outcome of its actions. An expectation system is introduced in the action selection mechanism, and it provides a mechanism to monitor, evaluate and report the effect of behavioral actions.

In a novel task, humans perform sequences of actions with a high degree of conscious awareness. But when a task is practiced, conscious control fades. We define automatization as an incidental learning process for skills that perform tasks with a predictable sequence, such as cycling and walking. Automatization grows stronger with practice. As a result a sequence is performed unconsciously or with less conscious involvement.

It is possible that automatic skills fail to perform a sequence. Failures could arise from a change in the internal state or in the environment. Failures tend to create novel situations and to become conscious contents. Deautomatization is a process of breaking up disrupted automatic skills to consciously accessible components. The details of automatization and deautomatization are covered in chapter 5.

The following questions are raised as a result of the research objectives to design and implement expectation, automatization and deautomatization mechanisms.

- How can expectation mechanism be specified, represented, and integrated in an action selection system?
- How and in what situations may expectations become consciously accessible?
- What are the conditions for automatization?
- How can we represent the skill learning process in the automatization mechanism?
- How does automatization affect conscious access?
- How are failure conditions detected in automatic skills?

- How does deautomatization break skills into consciously accessible components?

- What are the effects of the automatization mechanism on the interaction of "consciousness" and goal context system?

- Is it necessary to forget automatized skills? If so, how?

Consciousness has many functions (Baars, 1997). The expectation, automatization and deautomatization mechanisms will show two of the functions of consciousness: (a) the learning and adaptation function and (b) the error-detection and editing (of action plans) function.

1.5.2.3 Non-routine Problem Solving Mechanism

As we will discuss in chapter 6, a non-routine problem arises from an impasse or when an agent encounters a novel situation and does not have a readily available routine solution that can handle the novelty.

The research objective here is to provide a detailed design specification of a non-routine solving mechanism and to discuss how the mechanism integrates into IDA's action selection system. This objective raises the following questions.

- How can we detect impasse?

- What is the mechanism of deliberation for non-routine problems?

- Is it necessary to generate new goal structures or behavior streams? If so, how?
- Is it necessary to generate new goal contexts or behaviors? If so, how?

Problem solving is one of the functions of consciousness (Baars, 1997) and this mechanism will show how this role of "consciousness" is realized by mobilizing unconscious resources in multiple cognitive cycles.

1.5.2.4 Goals

The questions raised above lead to some of the following research goals.

- To develop a behavior network definition language (BNDL) so that behaviors, goals and goal hierarchies can be specified in this language. The BNDL can be considered as ontology for a behavior network or goal hierarchy system.
- To implement a compiler that converts a BNDL specification to computer executable objects.
- To add capabilities to behavior network to support: (i) variables, (ii) independent behavioral action feedback mechanism using the principle of reafference (Freeman, 1995), (iii) parametric control of priorities of goal structures that compete over extended time, and (iv) planning and subgoaling.
- To design IDA's action selection mechanism in such a way that it realizes the goal context system postulated by the global workspace theory. Goal context system (i) helps to evoke, shape and produce actions, (ii) deals with tasks that

extend over multiple "conscious" events, (iii) represents future states, and (iv) constrains stream of "consciousness."

- To implement from scratch the action selection mechanism with emphasis on distributed computation and the separation of domain independent from domain dependent architectural properties, generality, and reusability.

- To create a software framework so that IDA's action selection mechanism can be deployed in a new domain with a relatively less programming effort and relatively quickly.

- To test IDA's action selection mechanism by developing an agent system that deals with a warehouse robot domain. Doing so is to evaluate how well will such action controller performs intended real world domain tasks compared to human operators and to demonstrate the mechanism's feature of controlling priorities of competing goal contexts hierarchies.

- To design and implement automatization (incidental skill learning), deautomatization (making skill components accessible for consciousness) and expectation (feedback for behavioral actions) mechanisms. The skill learning mechanism uses John Jackson's pandemonium theory (Jackson, 1987).

- To design a non-routine problem solving mechanism so that impasses can have access to "consciousness" and then "consciously" mediated or deliberative processes try to generate goal structures that could handle the impasses. The solution generation is a meshing process (Glenberg, 1997) of mental resources that are recruited by "consciousness."

The IDA architecture aspires to model human cognition, mostly qualitatively. In the cognitive modeling end, we hope that our design and implementation decisions will generate hypotheses on how the mind may work.

1.6 Dissertation Structure and Guide

The dissertation has six chapters. In the second chapter, we discuss the IDA architecture with a brief description of the different modules and related base mechanisms. Here, we also discusses the cognitive cycle of IDA, which consists of the cognitive steps available in a single run from sensation to action. The action selection system is one of the critical components of IDA.

In the third chapter, we describe Maes' behavior network, which is the basis for the action selection mechanism. We point out the attractive representation and personalities of Maes' behavior network that make it a suitable control structure for sophisticated decision making agents.

In chapter 4, we discuss IDA's action selection mechanism (IASM) (Negatu & Franklin, 2002) in depth. We cover the details of goal structures (behavior streams) and the framework of their components, behaviors and goals. IASM is a motivated system, and we present drives as primary motivators by representing built-in agenda. IASM, based on Maes' behavior network (1989), introduces new control parameters: (i) importance – the degree of relevance to satisfy a drive compared to other competing drives, and (ii) discrimination-factor – the degree of limitation in passing activation energy among competency modules of competing goal context hierarchies. A drive with highest importance parameter passes the highest goal-end external energy and vice versa. For the discrimination-factor parameter, a value of zero allows free flow and a value of one allows no flow of activation and it could be tuned to any value in between. These two parameters can be tuned to control priorities of goal context hierarchies in the decision process of the IASM.

Here we describe how a behavior network module models the goal context hierarchy system postulated in global workspace theory. As such, behaviors are represented as logical units of competency in the behavior network, as well as a coalition of processors, which we call behavior codelets. We discuss how behavior codelets, underlying behaviors, implement facilities to sense and assert relevance, as well as execute actions of the behaviors they underlie. We discuss the implementation of an expectation system using codelets to provide independent feedback on behavioral actions.

In chapter 4, we also discuss how "consciousness" recruits goal context hierarchies and, in turn, how goal contexts influence what comes to "consciousness." This leads us to cover the process of "conscious" recruitment and instantiation of behavior streams, the unconscious competition and selection of dominant goal contexts, the execution of action, and the effect of action that may become a "conscious" content. In this chapter, we present some hypotheses that come out of design decisions for IASM. We continue our discussion on the implementation of IASM, which includes architecture, building blocks (java classes) and their algorithms, behavior network specification language, and on how to deploy IASM in a new domain.

We continue in chapter 4 by reporting about tests and results. The objective of the test is to demonstrate how implemented IDA's action selection and "consciousness" modules interact as per global workspace theory and as per the cognitive cycle of IDA becomes a control system for a robotic agent that operates in a warehouse domain. We use a Khepera robot system that needs to perform different and sometimes competing tasks: unloading, shelving, unshelving, and shipping of different items and moving to charging stations to charge its battery. Here we will have a GOMS analysis of the domain tasks of the robots; this enables us to evaluate the performance of the IDA as a robotic agent. We will also test the usability of the newly introduced behavior net parameters in controlling priority of competing goal context hierarchies. Our test in this chapter concludes by comparing the performance of the robotic agent with a human operator into senses: (i) how well the IASM performs in comparison to its design specification, and (ii) how well the IASM performs compared to a human operator doing the same task.

Many researchers (Brooks, 1986; Rosenblatt & Payton, 1989; Tyrrell, 1993; Blumberg', 1994) have developed behavior-based action selection architectures. We will briefly discuss such related works before we conclude chapter four.

Having laid down the action selection mechanism and its interaction with "consciousness," we proceed to chapter 5 to discuss automatization and deautomatization mechanisms. Automatization is incidental learning to build skills for well practiced procedural tasks. There are evidences for the fading of conscious control from automatized tasks. Our objective is to investigate how architectural modules or bodily parts learn the high degree of coordination that makes conscious control unnecessary. Deautomatization also happens when skills are disrupted or when automatized actions fail to produce expected outcome. We, as many others, argue that unconscious encoding, interpretation, and inferences are triggered by conscious content; and we also argue that automatization is related to attention systems. Here we characterize the type of tasks that can be automatized and what representation may be changeable by the automatization learning process.

Our discussion elaborates the background evidence and related concepts for automatization. We develop requirements for automatization and then proceed to discuss the details of the mechanism, including the encoding of coordination among action processors (behavior codelets) and the effect of automatization on attention systems. We then discuss the conditions for deautomatization and develop its mechanism. We conjecture that deautomatization mechanisms should guide intensive attention to the disruptions of skills so that conscious control can take over. We also hypothesize that a deautomatization effect has a relatively short time span. That is, deautomatization is not the unlearning of skills.

We test the automatization/deautomatization mechanism with a simple task with a clearly defined sequence of behaviors – walking. Our test has the objective of demonstrating how automatization builds and deautomatization reinstates conscious control when a failure arises during the execution of automatized task. Particularly, we want to show how attention or "conscious" access fades and performance improves as automatization develops; we also want to show "conscious" access to automatized task events happens due to the effect of deautomatization that arises from an introduction of a failure during execution of the automatized task.

In chapter 6 we start by defining the concept of a non-routine problem solving in general and in relation to the IDA domain. We then develop preliminary ideas for a non-routine problem solver mechanism including a possible way to detect non-routine problem situation in IDA. We discuss our design approach in computational terms. Particularly, we approach the issue of non-routine problem solving as a special deliberative goal context hierarchy – behavior stream. We design the mechanism to be a behavior stream called Non-Routine Problem Solving (NRPS) behavior stream. Our research objective here is to design and specify the NRPS behavior stream and explain how it deliberates in the solution search process. It implements what Shallice (1988) calls the Supervisory Attention System (SAS) generator. We explain the details of each competence module in the behavior stream. We discuss the process and its interaction with other modules using the steps in cognitive cycle of IDA (see chapter 2). We discuss how the process could or fail to generate solutions that may handle the novelty.

In chapter 7 we summarize the contributions this research makes to the field of understanding and devising cognitively inspired, integrated decision making mechanisms for intelligent autonomous agents. We also point out possible new directions in which the research can be furthered in the future.

There are 4 appendices. While Appendix A gives the XML formatted grammar of the behavior network definition language (BNDL) that we developed appendix B gives a condensed BNDL source code to control a robot that operate in warehouse domain. Appendix C discusses the specification of the non-routine problem solving behavior stream (see chapter 6) as well as the modified partial-order planning algorithm to be used by the stream's planner behavior. Appendix D gives the complete NGOMSL analysis, which is produced by QGOMS tool, for the warehouse domain robot task.

2 IDA Architecture and Mechanisms

IDA is a "conscious" software agent that implements a global workspace theory of consciousness. It is intended to model a broad range of human cognitive functions. Its architecture, shown in Figure 2-1, is comprised of a number of different modules, each devoted to particular cognitive processes such as perception, associative memory, episodic memory, action selection (decision making), learning, emotions, "consciousness," automatization, expectation, problem solving and metacognition. To realize these cognitive modules, the design and implementation of IDA is based on various "new-AI" mechanisms including behavior network (Maes, 1990), copycat architecture (Hofstader & Mitchell, 1994), sparse-distributed memory (Kanerva, 1988), pandemonium theory (Jackson, 1987), and classifier systems (Holland & Reitman, 1978). In the coming sections of this chapter, we briefly discuss each of the modules and related mechanisms that constitute the IDA architecture (Figure 2-1).

2.1 Codelets

Global workspace theory postulates that specialized processors are the basis for human cognition. In IDA, codelets, using Hofstadter's terminology (1994), correspond to the special processors. They are special purpose, relatively independent, simple agents. Each codelet is implemented as a small piece of code running as an independent thread. Codelets are specialized for simple tasks and mostly work in coalitions to underlie high-level constructs such as behaviors. They also work independently, to often play the role of demons waiting for a particular condition to happen and acting on it as per their specialization. Codelets do almost all the work, making IDA a multi-agent system in a single cognitive system and in the sense of Minsky's society of mind (1985). Codelets can be seen as demons in the sense of Jackson's (1987) pandemonium theory.

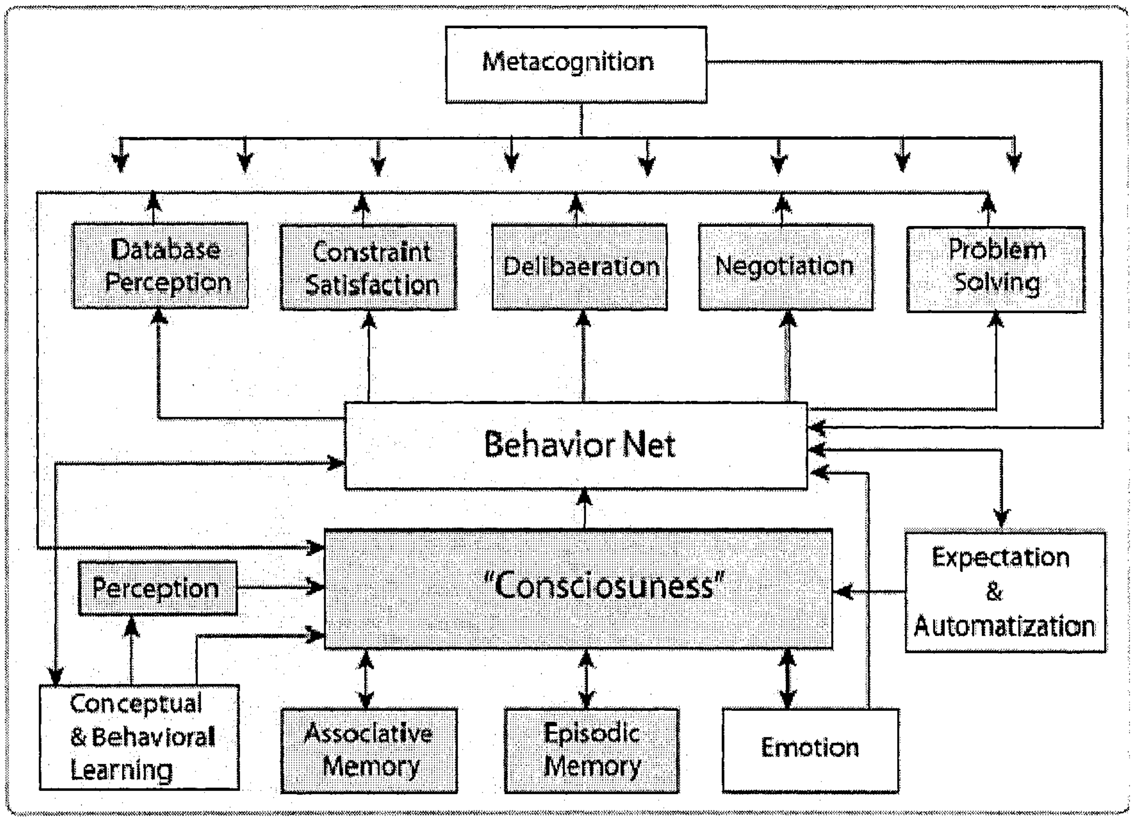

Figure 2-1: The IDA Architecture.

Codelets represent procedural knowledge that can execute primitive tasks. They also have an activation level, which represents the relevance of that knowledge in a particular context. Codelets can form associations with other codelets and a strong association may represent the formation of coalitions. Strong associations are also links for low-level, intra-codelet interactions. Association builds among co-active codelets as per the pandemonium theory.

There are many types of codelets, based on the specialization they have or the high-level module they underlie. These include perceptual codelets, attention codelets, information codelets, behavior codelets, and expectation codelets. We will encounter them in our discussion in the coming sections and chapters.

2.2 Perception

IDA senses strings of characters, not imbued with predefined meanings, but as primitive sensations as, for example, is the pattern of activity on the rods and cones of the retina in humans. These strings may come from email messages and from database records. The relative simplicity of the domain allows IDA's perception module (much like that of an earlier such agent (Zhang, Franklin, Olde, Wan, & Graesser, 1998) to employ analysis of surface features for natural language understanding (Allen, 1995). Its underlying mechanism constitutes a portion of the Copycat architecture (Mitchell, 1993; Hofstadter & Mitchell, 1994). The perceptual/conceptual knowledge base of IDA takes the form of a semantic net with activation passing called the slipnet. The name is taken from Copycat architecture. Nodes of the slipnet constitute the agent's perceptual symbols in the sense of Barsalou (1999). The slipnet embodies the perceptual contexts and some conceptual contexts from global workspace theory. A horde of perceptual codelets descend on an incoming message, looking for words or phrases (strings) that they recognize. When such are found, appropriate nodes in the slipnet are activated. This activation passes around the net until its dynamics settles. An idea-type node (or several) is selected by its high activation and the appropriate template(s) filled by codelets with selected items from the message. This is the process of understanding the message.

This process results in information being constructed by the agent for its own use (Franklin, 1995). This information is written on the workspace (short-term, working memory, not to be confused with Baars' global workspace). The workspace consists of registers that are associated with particular categories of information. Some of the registers constitute an interface (focus – as used by Kanerva, 1988) to IDA's long-term memory. Workspace contents decay over time and may be overwritten. Many of IDA's modules either write to the workspace, read from it, or both. For instance, as part of their act, perceptual codelets and behavior codelets write on the workspace; attention codelets watch for changes in the workspace and react to relevant changes.

So far we have described IDA's passive perception of incoming strings of characters. IDA also performs active perception in the form of reading external databases such as the sailor's personnel record, the job requisition list, the schedule of training classes, etc. This is the easier form of perception since there is an agreed upon protocol for each database. IDA may receive an e-mail message from a sailor asking for a new job assignment as his projected rotation date (PRD) is approaching. The perception module would recognize the sailor's name, social security number, and that the message is find-me-job idea type. These pieces of information are written on the workspace.

The underlying assumption for the design of the perception module is that much of human language understanding employs a combined bottom-up/top-down passing of activation through a hierarchical conceptual network, with the most abstract concepts in the middle. A node in the slipnet is a piece of declarative knowledge, and its activation level represents its degree of relevance in the situation. Each perceptual codelet represents some procedural knowledge. In the copycat mechanism, macro-level behaviors emerge from the activation spreading dynamics in the slipnet and actions by perceptual codelets. Surface features activate perceptual codelets, which, in turn, activate higher-level concepts in the slipnet.

2.3 Associative Memory

IDA employs sparse distributed memory (SDM) as its major long-term associative memory (Kanerva, 1988). SDM is a content addressable memory that, in many ways, is an ideal computational mechanism for use as a long-term associative memory. Being content addressable means that using part of their contents as a cue to retrieve items in memory, rather than having to know the item's address in memory.

The inner workings of SDM rely on large binary spaces, that is, spaces of bit vectors containing only zeros and ones. These binary vectors, called words, serve as both addresses and contents of the memory. The dimension of the space determines the richness of each word. These spaces are typically far too large to implement in any conceivable computer. Approximating the space uniformly with a possible number of actually implemented, *hard locations* surmounts this difficulty (Anwar, Dasgupta, & Franklin, 1999). The number of such hard locations determines the carrying capacity of the memory. Features are represented as one or more bits. Groups of features are concatenated to form a word. When writing a word to memory, a copy of the word is placed concurrently in all close enough hard locations. Each hard location contributes to the storage of several words. When reading a word, a close enough cue would reach all close enough hard locations and get some sort of aggregate or average out of them. Reading is not always successful. Depending on the cue and the previously written information, among other factors, convergence or divergence during a reading operation may occur. If convergence occurs, the pooled word will be the closest match (with abstraction) of the input reading cue.

The reading of memory in SDM is an iterative process. The cue is used as an address, whose content is read as a second address, and so on until convergence, that is, until subsequent contents look alike. Rehearsal is possible. An item can be written many times and, at each time, to a thousand locations, hence the "distributed" part of sparse distributed memory. A well-rehearsed item can be retrieved with smaller cues. Forgetting would tend to increase over time as a result of interference from other similar writes to memory.

Reads and writes to and from associative memory are accomplished through a gateway within the workspace called the focus. When any item is written to the workspace, another copy is written to the read registers of the focus. The contents of these read registers are then used as an address to query associative memory. The results of this query, that is, whatever IDA associates with this incoming information, are written into their own registers in the focus. This may include emotions and actions, which are experienced previously. Thus, associations with any incoming information, either from the outside world or from some part of IDA itself, are immediately available. Writes to associative memory are made later and will be described below.

2.4 Transient Episodic Memory

Episodic memory is a memory that can be traced to a specific event with features of place and time (Baddeley at el., 2001.) Events may record who did what to whom where and when. Episodic memory is part of the long-term memory system and, as such, is associative in nature, has relatively unlimited capacity, and allows retrieval of information after being out of consciousness for long period of time. The length of time could be in the range from few minutes to many years.

We humans have a transient episodic memory (TEM) with a decay rate measured in hours (Conway, 2001; Franklin et al, 2005). We can recall events of the current day in a relatively great detail – what we had for lunch, where we parked our cars, what we discussed with friends during coffee breaks, etc. But, such details of events will stay with us only for hours. IDA models TEM using a modified sparse distributed memory (SDM) with decay rates in hours (Ramamurthy et al, 2004). Information in the TEM that has not decayed is consolidated to the declarative memory. The modified SDM uses a ternary memory space ("0," "1," and "I-do-not-care.") This representation allows flexible partial cuing so that missing features are represented by "I-do-not-cares." Episodic memories must have binding-error detectors and integrators (Shastri, 2002), and IDA's TEM module provides the facility for these episodic functions. The modified SDM shows significant improvements over the original architecture. The improvement comes at the cost of larger space that is required to support the tertiary representation in the modified

SDM mechanism. The hypothesis here is that cognitive agents need transient episodic memory (TEM) to deal with the details of short term events in their dynamic environment and at regular time intervals, and un-decayed information in TEM is consolidated into declarative memory.

2.5 "Consciousness" Mechanism

The apparatus for producing "consciousness" consists of four components: a coalition manager, a spotlight controller, a broadcast manager, and a collection of attention codelets; also active codelets are organized in the style of the pandemonium theory (Bogner, 1999; Bogner, Ramamurthy, & Franklin, 2000). The working process of the global workspace is explained in the metaphor of a theater stage (Baars, 1997) and briefly discussed in section 1.2.3. The "consciousness" module components could be best understood as an implementation of some aspects of the processes involved in a theater stage.

2.5.1 Sports Arena

The pandemonium theory (Jackson, 1987) makes an analogy with sports arena, which is composed of stands and a playing field. In the stands, there are the inactive codelets. But each is watchful to a relevant condition that may happen anywhere in the arena and activates itself and joins the playing field when it detects that condition. Here, there is a difference with the pandemonium theory where demons in the stands are activated to join the playing field only by the strength of their association with demons that are currently active in the playing field.

Active codelets render their acts when they are in the playing field. Some are in coalitions and join the playing field together. An attention codelet could be in a coalition with information codelets in the playing field that would represent a snapshot of perceptual input and related information in the workspace.

2.5.2 Attention Codelets

Attention codelets recognize novel or problematic situations. Attention, which may be automatic or voluntary, is the control of access "consciousness." Attention codelets have the task of bringing information to "consciousness." Each attention codelet keeps a watchful eye out for a particular type of situation to occur that might call for "conscious" intervention. Upon encountering such a situation, the appropriate attention codelet will be associated with the small number of codelets that carry the information describing the situation, as well as some of the original data. This association should lead to the collection of this small number of information codelets, together with the attention codelet that collected them, becoming a coalition. Codelets also have activations. The attention codelet increases its activation as a function of how well it is suited to the current situation so that the coalition might compete for "consciousness" if such a coalition is formed.

2.5.3 Coalition Manager

The coalition manager is responsible for forming and tracking coalitions of codelets. Such coalitions are initiated on the basis of the mutual associations between the member codelets that are active in the playing field. An active codelet starts without associations and is in its own coalition. In the style of pandemonium theory, active codelets in the playing field build associations. Associations grow with co-activity and would result in coalitions with two or more codelets.

2.5.4 Spotlight Controller

The relative relevance of a coalition is based on the average activation level of its member codelets. The average activation decides the relative competitiveness of the coalition to become "conscious." The spotlight controller computes the average activation level of coalitions in the playing field. Then the spotlight controller chooses and shines on a coalition with the highest average activation level. IDA becomes "conscious" of the content of that coalition. A codelet's activation is influenced by the relevance of the high-level construct it underlies. A behavior (slipnet node) would influence activation of its behavior (perceptual) codelet. In the case of an attention codelet, its activation is influenced by its own action. Also, current emotional content may have direct/indirect influence on all types of codelets. Here, different coalitions that underlie different contexts (goal, perceptual) compete for access to "consciousness" and the spotlight controller declares the winner.

2.5.5 Broadcast Manager

Global workspace theory calls for the contents of "consciousness" to be broadcast to each of the codelets in the system. The broadcast manager accomplishes this, by compiling the information content of all the member codelets of the winner coalition chosen by the spotlight controller. Then it disseminates the compiled information as a single broadcast to all codelets. Each codelet decides whether relevant information is available in a broadcast.

2.6 Action Selection

IDA's action selection mechanism (IASM) (Negatu & Franklin, 2002) depends on the expansion of the idea in the behavior network (Maes, 1989; Song & Franklin, 2000). We cover the details of the mechanism in chapters 3 and 4. In chapter 3, we discuss Maes' behavior network and its properties. In chapter 4, we cover the complete detail of IDA's action selection system. In this section we briefly introduce the action selection mechanism.

IDA selects and executes its actions in the service of built-in drives. It may have several distinct drives operating in parallel, which vary in urgency as time passes and the environment changes. An active behavior network is composed of goal structures called behavior streams. Behavior streams have behavior and goal nodes. Behaviors and goals in a behavior network are interconnected with various links. Behaviors are typically mid-level actions, many depending on several behavior codelets for their execution. Goals are similar to behaviors except that their action is to conclude goal states under specified conditions. A behavior in IDA corresponds to a goal context in global workspace theory. A behavior looks very much like a production rule, having preconditions as well as additions and deletions. A behavior is distinguished from a production rule by the presence of an activation, a number intended to measure the behavior's relevance to both the current environment (external and internal) and its ability to help satisfy the various drives it serves. Action selection happens as a result of activation spreading in the behavior network.

IDA's behavior network acts in consort with her "consciousness" mechanism to select actions (Negatu & Franklin, 2002). A behavior stream is instantiated and become part of the active behavior network if an associated behavior codelet finds relevant information in a "conscious" broadcast. Behaviors, as goal contexts, do cooperate and compete with each other in the behavior network. The dynamics in a behavior network eventually selects a relevant behavior for action. A selected behavior activates its underlying behavior codelets, which, in turn, jump to the playing field and execute their intended actions. An action has been selected and carried out by means of collaboration between "consciousness" and the behavior network. Competition and cooperation of processors happen in the "consciousness" and action selection modules to produce the relevant next action of IDA. An action could be part of an automatic skill, consciously mediated procedure, or voluntary one.

The "consciousness" and the action selection modules play a central role. All the other modules of IDA we discuss next make heavy use of the "consciousness" and behavior network modules.

2.7 Constraint Satisfaction

IDA's constraint satisfaction module is designed to provide a numerical measure of the fitness of a particular job for a particular sailor (Kelemen, Liang, Kozma, & Franklin, 2002). Not only must IDA look out for the needs of the sailor, it must also see that the requirements for individual jobs are met and simultaneously adhere to the policies of the Navy. This is accomplished by use of a linear function with a term for each issue (particular need, requirement, and policy). The function of each term measures the sailor/job fitness with respect to that issue. The coefficient measures the importance of that issue with respect to the others, and there is also a multiplicative term to take care of the hard constraints.

2.8 Deliberation

Suppose a list of possible jobs is now present in the workspace, each with its fitness value. IDA must decide whether or not the sailor can detach from his current billet within the prescribed time window and arrive at the new job within its window, having taken leave time, proceed time, travel time, and sometimes training in-between. IDA's deliberation module creates and evaluates scenarios to make these decisions (Franklin, 2000a; Kondadadi & Franklin, 2001). In the deliberation process, scenarios are created by the interaction of the "consciousness" and behavior network modules, and by the action of attention and behavior codelets that are associated with scenario constituting events. Once one or more acceptable scenarios are in the workspace, IDA must finally decide on one or two jobs to offer the sailor. This is accomplished via voluntary action, an implementation of James' ideomotor theory (James, 1890; Franklin, 2000a; Kondadadi & Franklin, 2001) as prescribed by global workspace theory (Baars, 1988). Ideomotor theory suggests that conscious goals are impulsively carried out unless they are opposed by other conscious thoughts. A conscious goal selection activates a behavior stream that compiles a message, paragraph by paragraph, from scripts carried by behavior codelets. Their fields are filled in, using information from the workspace. All happens on the basis of the multiple-cycle interaction of the "consciousness" and behavior net modules.

2.9 Negotiation

After sending the offer message, IDA must deal with the sailor's response. Its negotiation module does this (Franklin, 2001). The sailor may accept an offered job, refuse the offered jobs, ask for another particular job, request a delay until the arrival of the next job requisition list, seek information about the requested job, etc. Each of these must be handled differently, many occasioning decision points for IDA. Negotiations can be quite extended, and IDA must occasionally assign the sailor to a job he or she has not accepted. The negotiation module makes heavy use of the other modules during its interchange with the sailor. This again happens with multiple-cycle interactions of the goal context and "consciousness" mechanisms.

2.10 Emotions

Oatley and Jenkins (1996) give a working definition of emotion as: "(1) An emotion is usually caused by a person consciously or unconsciously evaluating an event as relevant to a goal that is important; the emotion is felt as positive when a goal is advanced and negative when a goal is impeded. (2) The core of an emotion is readiness to act and the prompting of plans; an emotion gives priority for one or a few kinds of action to which it gives a sense of urgency – so it can interrupt, or compete with, alternative mental processes or actions. Different types of readiness create different outline relationships with others. (3) An emotion is usually experienced as a distinctive type of mental state, sometimes accompanied or followed by bodily changes, expressions, actions" (p.96).

To point out the importance of emotions for intelligent machine, Minsky (1985, sec. 16.1) said: *"The question is not whether intelligent machines can have emotions, but* *whether machines can be intelligent without any emotions. I suspect that once we give machines the ability to alter their own abilities we'll have to give them all sorts of complex checks and balances." Many other researchers also suggested that non-biological intelligent agents should have emotional capabilities* Frijda, 1986; *(*Sloman & Poli, 96; Cañamero, 1997; Picard, 1997; Velásquez, 1997). *IDA's conceptual model includes mechanisms for emotions (Franklin & McCaulley, 2002). IDA may feel guilty at not assigning a job for a sailor in time and be frustrated at not understanding a message.* Emotions in IDA are not in a separate module, but rather effect the actions of various codelets, particularly attention codelets. They also influence the activation passed by various drives (Demasio, 1994) to the action selection system. They influence the selection of and the emphasis on information content that needs to be written to the long-term associative memory. Thus, emotions effect essentially every action (McCauley & Franklin, 1998; McCauley, 1999; McCauley, Franklin, & Bogner, 2000). The emotion module of IDA is yet to be implemented.

2.11 Metacognition

Flavell (1976) defines metacognition as "knowledge concerning ones own cognitive processes and products or anything related to them ... active monitoring and consequent regulation and orchestration of these processes" (p. 252). There seems to be a broader agreement that metacognition should include knowledge of one's own cognitive processes and the ability to actively and consciously monitor, regulate and orchestrate them. In humans, monitoring, regulation, and orchestration can take the form of checking, planning, selecting, inferring, self-interrogation, introspection, interpretation of ongoing experience, or simply making judgments about what one knows or does not know to accomplish a task (Hacker, 1998). A metacognitive module has been implemented in our previous "conscious" software agent project - CMattie (Zhang et al., 1998). The implementation uses different mechanisms such as fuzzy controllers, classifier systems, and fuzzy classifier systems and actively, but unconsciously, monitors and tunes the global parameters of the behavior network module so that it could run more optimally. At the moment, the metacognitive module is part of IDA's conceptual model in a new guise that implements it directly within IDA's underlying cognitive cycle architecture, rather than with a different mechanism.

2.12 Learning

The IDA model has different learning and knowledge acquisitions mechanisms, which are integrated in its various modules (Ramamurthy et al., 1998a, 1998b, 2001; Negatu & Franklin, 1999).

There is a type of declarative learning in the long-term Associative Memory module. This learning happens as a result of storage of "conscious" contents in the associative memory. Every "conscious" broadcast is learned. Learning different items in the workspace (precepts, actions, and contexts) also happens in the associative memory module. The current focus of our research group is now shifted in addressing the learning capacity of IDA – to build a learning IDA (LIDA).

We humans develop skills from repetitively performed procedures or sequence of skills. Automatization, which will be the focus of our discussion in chapter 5, is a learning mechanism that enables IDA to acquire skills. The automatization mechanism is based on the pandemonium theory style organization (Jackson 1987) of codelets to encode sequences of activities in the playing field.

Furthermore, although at the early stage of design, IDA is intended to have mechanisms for conceptual learning (Ramamurthy et al., 1998a) and behavioral learning (Negatu & Franklin, 1999). Both the conceptual and behavioral learning mechanisms are based on the premises: (a) agents learn based on what they already know; and (b) agents approach every new situation in terms of a previously solved problem or in terms of old plans that worked in similar situation. They use a multi-strategy learning approach by gathering knowledge about differences of new entities with known and similar entities. The conceptual learning categorizes new concepts from knowledge of similar concepts. Then it carries conversations with humans to elicit knowledge that refines the description of the new concept. Similarly, the behavioral learning picks a likely applicable behavior stream for a new situation from behavior streams that are applicable to similar situations. Then it carries out conversations with humans to elicit knowledge that adapts or refines the picked behavior stream as a solution for the new situation. Both are an online learning type. The main motivation in the design of these mechanisms is to have faster learning, which is achieved by the effective use of the internal knowledge of IDA (exploiting analogy with known entities and past experience) and the knowledge of others (eliciting description about entities from humans). Successful learning could happen in one shot or with few cycles of conversation with humans.

2.13 Cognitive Cycle

Like any other agent system, IDA passes through multiple cycles of sensing, deciding and acting. This is of course a very course cycle and each could be refined differently by different cognitive models. IDA continuously iterates in a cycle that activates its many modules. The IDA model suggests a cognitive cycle with nine major steps. IDA's cognitive cycle is discussed by Baars and Franklin (2003), and it is reproduced in table 2-1. Steps 1-3 involve perceptual, long-term associative memory and episodic memory modules, as well as the working memory of IDA. The steps 4-9 primarily involve the interaction of the "conscious" and action selection modules (Negatu & Franklin, 2002). See Chapter 4 for the details.

The cognitive cycles may overlap or run in parallel except in the case of "conscious" broadcast (step 5), which is the limited-capacity and inline with the global workspace theory. The conjecture is that a full cognitive cycle might take approximately 200 ms (Baars & Franklin, 2003). As we will discuss in chapter 5, performing well practiced tasks might be performed unconsciously with automatized action sequences or skills. The automatization effect is to shorten the cognitive cycle by bypassing the "conscious" recruitment of resources. When skills are in play, currently active behavior related codelets could directly recruit the underlying codelets of the next behavior in a sequence. Such low-level, unconscious inter-codelet association happens via associations build as a result of practice. The overlapping of cognitive cycles and role of automatization would enhance the parallelism and avoid the limited-capacity in performing automatized tasks.

Table 2-1: Steps in the cognitive cycle of IDA (from Baars & Franklin, 2003).

Cognitive step	*Details of the cognitive step*
1. Perception	Sensory stimuli, external or internal, are received and interpreted by perception creating meaning. Note that this stage is unconscious. *Early perception:* Input arrives through senses. Specialized perception codelets descend on the input. Those that find features relevant to their specialty activate appropriate nodes in the slipnet (a semantic net with activation). *Chunk perception:* Activation passes from node to node in the slipnet. The slipnet stabilizes, bringing about the convergence of streams from different senses and chunking bits of meaning into larger chunks. These larger chunks, represented by meaning nodes in the slipnet, constitute the percept.
2. Percept to Preconscious Buffer	The percept, including some of the data plus the meaning, is stored in preconscious buffers of IDA's working memory. These buffers may involve visuo-spatial, phonological, and other kinds of information.
3. *Local Associations*	Using the incoming percept and the residual contents of the preconscious buffers as cues including emotional content, local associations are automatically retrieved from transient episodic memory and from long-term associative memory. The contents of the preconscious buffers together with the retrieved local associations from transient episodic memory and long term associative memory roughly correspond to Ericsson and Kintsch's long-term working memory (1995) and Baddeley's episodic buffer (2000).
4. *Competition for Consciousness*	Attention codelets, whose job it is to bring relevant, urgent, or insistent events to consciousness, view long-term working memory. Some of them gather information, form coalitions and actively compete for access to consciousness. The competition may also include attention codelets from a recent previous cycle.
5. Conscious Broadcast	A coalition of codelets, typically an attention codelet and its covey of related information codelets carrying content, gains access to the global workspace and has its contents broadcast. This broadcast is hypothesized to correspond to phenomenal consciousness. The contents of perceptual memory are updated in light of the current contents of consciousness, including feelings/emotions, as well as objects, and relations. The current contents of consciousness are also stored in transient episodic memory as events. At recurring times, not part of a cognitive cycle, the contents of transient episodic memory are consolidated into long-term declarative memory (Shastri, 2002).
6. Recruitment of	Relevant behavior codelets respond to the conscious broadcast. These

Resources	are typically codelets whose variables can be bound from information in the conscious broadcast. If the successful attention codelet was an expectation codelet calling attention to an unexpected result from a previous action, the responding codelets may be those that can help to rectify the unexpected situation. Thus consciousness solves the relevancy problem in recruiting resources.
7. Setting Goal Context Hierarchy	The recruited processors use the contents of consciousness, *including feelings/emotions*, to instantiate new goal context hierarchies, bind their variables, and increase their activation. It's here that feelings/emotions directly affect motivation. They determine which terminal goal contexts receive activation and how much.
8. Action Chosen	The behavior net chooses a single behavior (goal context), perhaps from a just instantiated behavior stream or possibly from a previously active stream. This selection is heavily influenced by activation passed to various behaviors influenced by the primary drives or emotions. The choice is also affected by the current situation, external and internal conditions, by the relationship between the behaviors, and by the residual activation values of various behaviors.
9. Action Taken	The execution of a behavior (goal context) results in the behavior codelets performing their specialized tasks, which may have external or internal consequences. This is IDA taking an action. The acting codelets also include an expectation codelet (see Step 6) whose task it is to monitor the action and to try and bring to consciousness any failure in the expected results.

Behavior codelets, once recruited (step 6), instantiate relevant behavior streams or set goal context hierarchies (step 7); i.e., if they are not already instantiated. The instantiation of the behavior steam is the process of choosing a routine solution for a possibly novel content. Some novel situations could not be handled by known solutions. Failing to have a routine solution in step 7 will trigger a non-routine problem solving process, which is the topic of our discussion in chapter 6. The non-routine problem solving mechanism will try to create new behavior streams as solutions to novel situations.

The effect of the emotion module is distributed, and its direct and indirect influence plays a central role in perception, memory, and "consciousness" and action selection. The other modules (deliberation, negotiation, and constraint satisfaction) operate in the service of the action selection mechanism, and they do not have separate roles in the cognitive cycle.

2.14 Conclusion

The IDA model is a cognitive architecture that integrates many cognitive processes using different modules that implement different mechanisms. This makes IDA different from other cognitive architectures such as ACT-R (Anderson & Lebiere, 1998) and SOAR (Laird et al., 1987), which are based on the unified theory of cognition (Newell, 1990).

We can characterize IDA's architecture in several ways. First, like many other models (ACT-R, DUAL), IDA is a hybrid system (symbolic/connectionist); particularly the copycat mechanism in the perception module and the behavior network, where there are symbolic schemes as nodes and activation spreading among them. Second, IDA's many modules, which describe the high-level cognitive constructs, may hide the fact that the low-level processes can have flexible integration or coordination. That is, everything in IDA is done by codelets (simple processes). As a coalition, they underlie many of the high-level computational constructs. Thus IDA's architecture is conceptually in agreement with the Minsky's Society of Mind (1986).

Sloman (2001) has proposed H-CogAff: a human-like cognitive architecture. In the H-CogAff schema, there are reactive mechanisms, deliberative reasoning, and meta-management layers. Each of these processing layers has a component in the perceptual and motor systems. Then the schema is a three by three architectural components, each with different sorts of functionality. IDA seems to realize the H-CogAff architectural framework very well. Although, not explicitly shown in the architecture (Figure 2-1), IDA can support a reactive mechanism; a subliminal perception can prime action operators (behavior codelets) in the behavior net that can produce reactive response. Among others, deliberation and negotiation in IDA are cognitive tasks that fall in the deliberative processing of H-CogAff. IDA's metacognitive module corresponds to the meta-management layer. The deliberative processes in IDA are served by actions controlled by behaviors in the behavior network, and behaviors are primed by contextualized perceptual contents that pass through consciousness. Likewise, the metacognition needs perception and action to provide its services like monitoring and regulation of self. So, IDA fits well in the schema of H-CogAff.

A cognitive architecture is a theory about the fixed computational mechanisms and structure of cognition (Pylyshyn, 1984; Newell, 1990; Anderson & Lebiere, 1998). The different modules, among which the "consciousness" and action selection modules play the central role, constitute the fixed structures of the IDA architecture. A cognitive architecture should support cognitive behavior by processing variable cognitive contents. IDA's cognitive cycle is the process that runs on the architecture. The cognitive cycle is inherent to IDA's architecture and thus a constant. The cycle defines our hypothesis on how humans, in order to exhibit their cognitive behavior, use a common or fixed process to manipulate cognitive contents over the various architectural components of IDA. The IDA architecture and its cognitive cycle forward a theory of what is common to the many of the human cognitive behaviors.

3 MAES' Behavior network

In the last two decades, many AI researchers (Brooks, 1986; Rosenblatt and Payton, 1989; Tyrrell, 1993; Blumberg, 1994; and others) have developed behavior-based action selection mechanisms. But our focus in this chapter is on Maes' (1990) behavior network, which is the basis for IDA's action selection system. Maes' action selection mechanism (MASM) is constructed as a society of behaviors or competence modules organized in a network. The model is closely allied to Minsky's (1985), Ornstein's (1986), and Jackson's (1987) idea of multiplicity of mind. That is, each agent unit is fairly simple, mindless, has its own specific competence and interacts with others via the spreading of activation to build an intelligent agent system. MASM has a predefined network with a fixed set of competence modules, but it has a dynamic and distributed action selection mechanism. MASM exhibits planning capabilities that emerge out of the activation spreading dynamics that covers the entire network of competence modules. A sequence of behaviors that transform the current state into the goal state becomes highly activated as a result of relevance to current situation (forward spreading) and to the targeted goal state (backward spreading). Plan in behavior network dynamic is not explicit structure. Rather plan is expressed as an "intention" to take an action based on the high activation level of behaviors in a sequence. The model integrates symbolic and connectionist characteristics. It is a connectionist computational model on a symbolically structured representation.

3.1 The mechanism

Each behavior by itself can be viewed as a production rule or an operator in an AI planning system with preconditions and actions (delete & add lists) with the addition of an activation level. More formally, a behavior is described as a tuple $(c_i, a_i, d_i, \alpha_i)$, where, c_i is a list of preconditions that must be satisfied for the behavior to become executable; a_i and d_i represent the post-condition or expected outcome after the behavior has become active. The behavior promises that after its action is performed, it sets the propositions in the add list and unsets the propositions in the delete list. α_i is the activation level of the behavior. Besides the tuple, each behavior may have an executable code that realizes the action of the behavior. The code is not explicit – it could be hard wired to control motor action, to make inferences, or any other operation.

Behaviors are connected in a network with causal links. Three types of links, successor, predecessor, and conflicter links, are constructed based on the shared conditions/propositions of behaviors in the precondition, delete and add lists. Formally, the links are defined as follow.

1. *Successor links* – considering behaviors X and Y and every proposition p, if p is in add list of X and p is in precondition list of Y, then there is a successor link from X to Y corresponding to P (X can help Y to be executable); more than one successor link may exist between X and Y. Formally: given $X = (c_x, a_x, d_x, \alpha_x)$

and $Y = (c_y, a_y, d_y, \alpha_y)$, there exists a successor link from X to Y for every proposition $p \in a_x \cap c_y$.

2. *Predecessor links* – for every successor link from Y to X, there is a predecessor link from X to Y. Formally: given $X = (c_x, a_x, d_x, \alpha_x)$ and $Y = (c_y, a_y, d_y, \alpha_y)$, there exists a predecessor link from X to Y for every proposition $p \in c_x \cap a_y$.

3. *Conflicter links* - considering behaviors X and Y and every proposition p, if p is in delete list of Y and p is in precondition list of X, then there is a conflicter link from X to Y corresponding to P. Formally: given $X = (c_x, a_x, d_x, \alpha_x)$ and $Y = (c_y, a_y, d_y, \alpha_y)$, there exists a conflicter link from X to Y for every proposition $p \in c_x \cap d_y$.

The spreading of activation among behaviors and injection of new activation energy into the network is determined by the state of the environment, the global goals, and the global parameters θ, π, γ, ϕ, and δ.

The global parameters are: a) θ is the threshold for becoming active, b) π is the mean level of activation in the network, c) γ is the amount of activation energy a global goal injects into the network, d) ϕ is the amount of activation energy injected by each proposition observed to be true in the environment, and e) δ is the amount of activation energy taken away by each protected goal – a satisfied goal that needs to remain satisfied.

- Environmental excitation – if a) proposition p, in the state of environment, is true, b) p is in precondition list of behavior X (i.e., X is partially relevant to the environment), then an environmental activation energy is injected. Amount of activation is $\frac{\phi}{NM}$, where N is the number of propositions in the precondition list of X, and M is the number of behaviors that have or match proposition p in their precondition list.

- Goal excitation – if a) global goal G is active (once-only and periodical goals are not active all the time while permanent goals are continuously active), b) goal G is in the add list of behavior X (i.e., X is likely to satisfy G), then goal (motivational) activation energy is injected. Amount of activation is $\frac{\gamma}{NM}$, where N is the number of propositions in the add list of X, and M is the number of behaviors that can achieve the goal G.

- Protected Goal inhibition – an active global goal G that has already been achieved and should be protected will take away activation energy from the behavior X that has G in its delete list. This is an external inhibition that takes away activation energy from the network. Amount of activation taken away is $\frac{\delta}{NM}$, where N is the number of propositions in the delete list of X, and M is the number of behaviors that can undo the goal G.

Aside from the external activation spreading, behaviors excite and inhibit each other along the three links as follows:

- Excite successors – an executable behavior X sends activation (a fraction of its own activation level) forward to behavior Y via the successor links if corresponding proposition $p \in a_x \cap c_y$ is false. Amount of activation is $\frac{\alpha_x \phi}{\gamma NM}$, where N is the number of propositions in the precondition list of Y, and M is the number of behaviors that have or match proposition p in their precondition list.

- Excite predecessors – a behavior X that is not executable will send activation (a fraction of its own activation level) backward to behavior Y via predecessor links if the corresponding proposition $p \in c_x \cap a_y$ is false. Amount of activation is $\frac{\alpha_x}{NM}$, where N is the number of propositions in the add list of Y, and M is the number of behaviors that achieve or have p in their add list.

- Inhibit conflicters – Every behavior X, whether it is executable or not, will send inhibition (a fraction of its own activation) via a conflicter link if the corresponding proposition $p \in c_x \cap d_y$ is true. A behavior protects its executability by inhibiting those that undo its satisfied precondition. Amount of activation is up to $\frac{\alpha_x \delta}{\gamma NM}$, where N is the number of propositions in the delete list of Y, and M is the number of behaviors that undo or have proposition p in their delete list. Note that X takes away from Y only if $\alpha_x > \alpha_y$.

The algorithm, shown in figure 3-1, first spreads external activation from the environment and global goals. Next activation spreading takes place among behaviors and then the activation level is normalized. At this point, competition takes place among behaviors. This satisfies two conditions: (i) executable and (ii) activation level over the threshold. The winner is the one with the highest activation level. The winning behavior, if there is one, become active and its activation level is set to zero and the threshold is set to the initial value. If none of the behaviors satisfy the two conditions, the threshold is reduced by 10%, and the process is repeated.

The global algorithm performs as follow at each time step in the loop:

1. The environment and global goals inject activation to the behaviors and protected global goals take away activation from behaviors.
2. Activation is spread among the behaviors using successor, predecessor and conflicter links.
3. Activation levels are normalized at the behaviors so that the average activation level in the network remains constant.
4. Behavior is active if it fulfils the conditions: (i) it is executable; (ii) its activation level is over threshold; (iii) it has the highest activation level among the others that satisfy conditions (i) and (ii).
5. A selected behavior, after it finished its action will reset its activation level to zero and threshold level is reset to default.
6. If no behavior is selected, reduce the threshold by 10%.

Figure 3-1: Maes' Behavior Net algorithm.

Aside from the global parameters, the amount of activation or inhibition is determined by two more factors: (i) the cardinality of the destination proposition list (N) and (ii) the size of connected components (M) at the point of spreading or inhibition. The first factor (N) removes the bias of stronger activation/inhibition energy that spreads towards behaviors with a larger size of the relevant proposition list. By default, the preconditions are in conjunctive normal form. A disjunction at precondition can be represented by two behaviors that can make the precondition proposition true. That is, behaviors that achieve the same goal or precondition state represent disjunctive choice points. The second factor (M) divides the activation/inhibition energy among the affected behaviors. Such factors are incorporated in spreading activation/inhibition energy from the environment, the goals, and the behaviors.

3.2 Properties of Maes' model

As a connectionist system over symbolic modules, this model inherits important properties from both worlds. From its connectionism side, it shows properties that include parallelism, fault tolerance or robustness, sophisticated matching, and emergence of global behavior from simple local decisions and interactions. From its symbolic side, it has properties such as symbolic programmability including prewiring with meaningful modules, accessibility of modules for explanation, and ease of integration of large and fine grain knowledge units. The dynamic flow of activation/inhibition in the system chooses a sequence of behaviors to act and eventually satisfy a goal – the dynamics in the mechanism plans without explicit representation and without AI type symbolic search process. All these are important properties in our agent system.

In addition, the global parameters provide the control to tune the mechanism in different, but equally important, opposing qualities such as adaptivity versus thoughtfulness, quickness and reactivity versus rationality. We will briefly discuss all such qualities.

Goal-orientedness is achieved by the backward spreading of activation from global goals to sub-goaling modules. Global goals inject γ activation to their sub-goaling modules, and in turn the sub-goaling modules inject activation via the predecessor links to their sub-goaling modules, and so on. The back propagation dynamics favors competence modules (a) that contribute to more than one goal, (b) that are topologically "closest" to the goal, and (c) that have less competition. The system can be more or less goal oriented by changing the γ to ϕ ratio. The higher the ratio, the more goal-oriented the system is. One extreme is if $\phi = 0$, then it is a complete goal-orientedness or pure backward chaining. But, at the same time, as the system becomes less *opportunistic*, less *reactive*; it is also *slow* as it does not take the state of environment in to consideration. So the γ to ϕ ratio should be tuned to the right point to the given domain.

The forward activation spreading from the state of the environment activates modules that are relevant to the situation. When they become executable, these modules, in turn, spread activation to other modules via successor links. By increasing ϕ to γ ratio, the system becomes more *situation relevant*, which also means that the system is able to exploit opportunities presented in the environment, and is able to reach speedy action selection with the bias coming from the environment and, as such, becoming more reactive.

The mechanism is *adaptive*. For dynamically situated agent, unforeseen situations can arise. The environment, as well as its active goals, may change from time to time. Also, some behaviors may be disabled. The mechanism continuously reevaluates the current situation and can easily adapt to changes. The network is also implicitly *biased to on-going plans*. The mechanism preserves the history of activation spreading after each action, and there is a bias towards firing patterns that represent the on-going action plan. Besides, more *bias to on-going plans* and less *adaptability* can be obtained by choosing γ (effect of goals) and ϕ (effect of environment) to be relatively smaller than π (mean activation level).

The network also has two more features with trade-off between them: *thoughtfulness* and *speed*. When the system is thoughtful, the activation spreading dynamics go on for some time before an action is selected. This is done by increasing the threshold θ. To the contrary, by keeping the value of θ lower, the network can select an action *faster*. Decisions may rest at local maxima and there is no looking forward in time for possible better action. As a result, the system becomes less thoughtful, less goal-oriented, less situation-oriented, less biased to ongoing plans. A good balance should be found in choosing θ in the given domain. Typically, speed or less thoughtfulness may be needed in a domain with a rapidly changing environment.

3.3 Conclusion

The Maes' behavior network is particularly interesting as an action selection system since it provides parameters to mediate between adaptivity and persistence, speed/reactivity and thoughtfulness/rationality, and goal-orientedness and situation-orientedness. The parameters provide external control to these action selection characteristics in the behavior network. Maes setup different behavior networks in toy domains to test the effects of the parameters in her algorithm and reported results. The results show effective controls on goal-orientedness, situation relevance and opportunistic behavior, adaptivity, bias to on going plans or persistence, avoidance of goal conflicts, thoughtfulness, and speed by tuning the different global parameters of behavior networks.

In a sequence of actions, goal-relevance of a behavior is obtained via input from the goals and the backward-spreading of activation. Situation relevance and opportunism are obtained via the input of the current environmental state the forward spreading of activation. Conflicting goals are accounted via inhibition by the protected goals and inhibition among conflicting competence modules. "Thinking" long enough (the threshold is high enough) avoids local maxima in the action selection and as a result the behavior network can emerge towards the optimal activity pattern. The mechanism has an implicit bias towards on going plans. The main reason is that activation levels are not reinitialized every time a competence module is activated or the remains of the past activation patterns contribute in selecting the next action.

Maes points out three main limitations in her mechanism: (i) it does not support classical variables and variable passing, (ii) it does not maintain a record of its past "search" or action path and as such, although rarely, loops in the action selection may emerge, and (iii) there is no a formal way to select values for the global parameters that can produce desired action selection characteristics in a specific application. The parameter selection is domain-dependent in the sense that not only different domains require different characteristics (goal-orientedness, reactivity, etc.) but also the characteristics are affected by the size and structure (long, wide) of the behavior network.

Tyrrell (1993) raised some concerns on Maes' behavior network algorithm. He points out that scheme of dividing the distribution of activation energy will always end up unfairly biasing one behavior over the other based on the number of fan-ins or fan-outs associated with each predecessor and conflicter causal relationship behaviors have with other competence modules. If backward spreading activation energy is divided evenly among all modules that satisfy a precondition, modules with pre-conditions that many other modules can satisfy are unfairly disadvantaged. On the other hand, if energy is given to each module without dividing by the number of associated energy receiving modules, modules that are unique in satisfying a goal are unfairly advantaged. Tyrrell concluded that no satisfactory set of division rules exists for the predecessor and conflicter connections. He also criticizes Maes' algorithm on some more grounds such as lack of real-valued representation and loss of relevant information due to binary representation of the environmental state (impoverished perception), and lack of combining preferences in the mechanism. The criticisms are valid if we take Maes' mechanism as a complete action selection system. However, they are not fundamental problems to Maes' approach. Her algorithm has been augmented (with added complexity to maintain and pass additional information) by others to address these and other concerns (Rhodes, 1996; Decugis & Ferber, 1998; Dorer, 1999).

Maes' action selection mechanism has a number of promising qualities: it is robust and adaptable; it is capable of pursuing multiple goals and show goal-orientated behavior, whilst still responding to the opportunities and threats encountered in a dynamic and unpredictable environment; it provides external control of its action selection properties via the tuning of a small number of global parameters; and it supports interruption of a running action plan by a new and relatively more "urgent" action plan. No hierarchical and no global control exists in the Maes' behavior network. The dynamics of spreading of activation decide the relevant (not necessarily optimal) action. The dynamics also implement a simple form of planning that is unlike the classical AI planning.

As we will discuss in chapter 4, IDA's action selection mechanism is based on Maes' mechanism with important addition (Negatu & Franklin, 2002). We incorporate classical variables and variable passing. We also add an expectation mechanism that monitors and provides feedback on the action of each behavior. More importantly, our action selection module implements the goal-context hierarchy system of global workspace theory and as such is coupled or integrated to the "consciousness" mechanism of IDA. As we will see in the coming chapters, our action selection system is predisposed for multiple strategy and sophisticated control of action – automatic, intentional, deliberative and consciously-mediated control of action.

4 Action Selection system

We will begin this chapter by describing the 'action selection problem.' We will discuss the computational components of IDA's action selection mechanism. Then we will see how the action selection module fits in to Global Workspace theory and how it interacts with the other computational modules of IDA. We will explain how action selection takes place in IDA's cognitive cycle. We will also discuss the implementation details and test results, as well as related works.

4.1 Introduction

An *autonomous agent* (Franklin & Graesser 1997) is a system situated in and part of an environment, which senses that environment and acts on it, over time, in pursuit of its own agenda. In biological agents, this agenda arises from drives that evolve over generations. Drives are encoded to enhance genetic fitness – survive, thrive and reproduce. Many animals exhibit fairly common behaviors, and Dewsbury (1978) mentions the following: locomotion, ingestion (including feeding, drinking, breathing), thermoregulation, seeking of shelter, avoidance of predators (including concealment, warning of conspecifics, escape, fighting), sleep, body maintenance (cleaning), elimination (urination & defecation), exploration (searching), play, use of tools, reproduction (including courtship, mating, care of eggs and care of young) and social behavior (including dominance, territoriality, aggression, and social facilitation). An animal with such primary behaviors (agenda) perceives its environment and its internal state, and needs to decide on the appropriate next action in service of one or more items in its agenda. Every moment, the animal, an autonomous agent, with multiple goals and behaviors, faces the *action selection problem* – the problem of deciding the next rational action, and to do so, it uses its knowledge and its reasoning capability in the service of its agenda. An action selection mechanism is a computational system that can handle the action selection problem by specifying the method of choosing an action.

Franklin (1995) posed the question "what are minds for?" and then answered this question as follows.

> *Any (autonomous) agent, be it a human or a thermostat, has a single, overriding concern – what to do next. And that's what minds are for – to address this concern by producing an appropriate action. In engineering terminology, minds are control systems. They balance all the mechanisms so that appropriate actions are continually generated in a timely fashion. … And all our internal jabber – feelings, beliefs, desires – subserve this single function of choosing what to do next (pp. 233).*

That is, the mind is best viewed as a control structure for an autonomous agent. Its overarching continual task is to produce the next action. Thus, every agent must repeatedly solve the action selection problem. In the design and implementation of software agents, modeling, design and implementation of an action selection mechanism (ASM) plays the central role. Primary motivation is essential. An ASM chooses the next action so the agent can satisfy one or more of its drives. So, what are the criteria of an

ASM for a "conscious" software agent? In general, an ASM should address many problems including: (a) *relevance* – how well an agent produces the most appropriate action to a given environment and its internal state, (b) *persistence* – how well an agent is committed in completing a course of action underway, and (c) *intentionality* – how well the state of mind has some level of commitment towards achieving a goal. The IDA model is a distributed computational architecture. To allow an easy functional and representational integration of the action selection system into the IDA architecture, it is a natural requirement for the action selection mechanism to be a distributed system.

IDA is a sophisticated agent and has many types of direct action control (automatized, voluntary, and consciously mediated). Also, IDA's conceptual model has a meta-cognitive system, which may automatically fine tune the behavior (situation-directed versus goal-directed or persistent versus opportunist, etc.) of the action selection system. So, IDA's action selection system should provide a relatively extended access to external controls. Maes' (1989, 1990, 1991) behavior network is a complex action selection mechanism, which is distributed, recurrent, and non-hierarchical. Maes' model provides system control with some global parameters. IDA's action selection system is based on, and extends, this model, which we have discussed it in more detail in Chapter 3.

4.2 Action Selection Mechanism

In this section we will discuss IDA's action selection mechanism that is based on Maes' behavior network model. It adds a behavior instantiation mechanism. IDA is a distributed computational architecture in the context of multiplicity of mind - all computational processing is done by small, mindless modules, which we call "codelets," each with its own agenda and very limited or local interaction. Maes' model, besides the many desirable properties, asserts two hypotheses that make a good base to our action selection system.

- A mutual activation/inhibition of competence modules produces a global system to select a 'good enough' action.
- No explicit global control/arbitration module is needed; global control emerges from local decisions (which ones to activate and which ones to inhibit) of competence modules.

Maes' action selection mechanism (MASM), also referred to as a behavior net, has received considerable attention. Tyrell (1993) has investigated MASM and reported on its strengths and limitations. Dorer (1999), and Decugis and Ferber (1998) introduced a variation of MASM by modifying the domain representation or the network structure.

Neither MASM nor any of the above variations incorporate variables or variable passing in the behavior network; i.e., there must be a distinct behavior defined for each simple action. As a consequence, for complex domains, the behavior network must have a prohibitively large number of behaviors. Maes (1990) argues that the introduction of variables will destroy the merits in her algorithm. Still, to apply the behavior net to complex domains, the introduction of variables is a necessity. We do so.

Rhodes (1995) proposed a variation of MASM that introduces variables using the indexical-functional aspects (Agre & Chapman, 1987). He defines real world functions in the task domain that help the agent to relate objects in its environment. This approach reduces design time considerations if the domain task can be captured in the behavior network with a relatively small number of functions compared to the number of objects in the environment. In the worst case, the number of functions could approach the number of items.

We added the capability to handle variables to the MASM by having instantiation of behaviors as they are needed. The introduction of classical variables to behavior network makes the selection control mechanism a hybrid (symbolic/connectionist) system. Song and Franklin (2000) presented this approach, and our work is based on this variation of MASM with some additional extensions. Here are the most important additions:

- Our work provides a more sophisticated hierarchical goal context system for our "conscious" agent.

- Our work is implemented so that it could serve as a developmental tool for a general-purpose cognitive agent; i.e., easy to deploy the system for different domains.

- Ours is a completely new implementation with true distributed parallelism where all mechanism components, including links, are independent threads. This adds to the adherence of our implementation to the modeling of distributed computational agent systems.

- Our work provides a flexible base action selection system for various cognitive processes. Streams, as partial action plans, can be organized hierarchically, which will allow it to handle important classes of cognitive functions such as non-routine problem solving, procedure automatization, and learning of new behavior streams (Negatu & Franklin, 1999).

- Our work introduces two new global parameters that could be tuned with the other parameters to control priority of competing goal hierarchies in the dynamics of the behavior net mechanism.

In the following sub-sections we will discuss the different components of the mechanism. We will start with the motivating module.

4.2.1 Primary Motivator

Motivation is a control state that moves the action selection system towards a desired physical/mental state. It is a module with dispositional influence to determine internal or external action, and which subsumes desires, goals, intentions, and wishes (Wright, 1997). In the moving towards a desired state, motivators perform three functions (Canamero, 1997; Kandel et al., 1995): (1) *directing* – they guide behavior towards satisfying a particular goal, (2) *activating* – they instigate an agent into action, and (3) *organizing* – they bring together individual behaviors into a coherent, goal-directed response.

4.2.1.1 Drives

The concept of drive is most commonly defined as an intervening variable and associated with the work of Hull (1943). A set of stimuli will determine the importance/urgency of performing a particular stream of actions. Drive can be seen as the combined activation effect of the many stimuli denoted as a single intervening variable value. The drive value for each behavior gives the total motivation of performing that behavior. The stronger the drive, the stronger the motivation, and the more likelihood an agent performs the behavior.

IDA's current motivator module is implemented as a drive system. Drives are built-in or evolved-in primary and internal motivators. All actions are chosen in order to satisfy one or more drives and a drive may be satisfied by different goals. While drives are invariant except for changes in intensity over time, the goals that satisfy them vary along with the internal state of the agent and with the environment.

A drive in IDA has an *importance* parameter (τ) that denotes its relative significance compared to other drives. The importance is a real value in $(0,1]$. A drive with an importance value of 1 spreads the highest level of motivational activation to its underlying goals. If a drive has an instantiated goal structure hooked to it, the activation energy that the drive spreads to the goal structure is weighted by the importance value. Drives take the role of global goals in Maes' model and, as such, they inject the goal-directing activation energy to instantiated goal structures. The amount of activation energy injected to a satisfying goal/behavior is $\frac{\tau\gamma}{NM}$, where γ is the amount of activation energy a global goal injects into the behavior network (see section 3.1), N is the number of propositions in the add list of the goal/behavior, and M is the number of goals/behaviors in the active network that directly satisfy the drive.

Hull (1943) proposed a general drive mechanism in which 'drive strength' is computed as a product of 'habit strength' and 'stimulus strength'; where habit strength is the result of learning from past experience and stimulus strength is some combination of relevant internal and external stimuli. It is the author's view that habit strength, for the most part, is evolutionarily learned or built-in. Stimulus strength modulates the intensity of the drive to swing up and down; a sense of hunger increases the drive for getting food and eating and, further, the perception of food intensifies the drive for eating. In the conceptual model of IDA, drives are realized as feelings. Feelings such as hunger and thirst are active, internal stimuli that implement the associated drives. That is, IDA's drives are directly based on some of the internal stimuli and primary in the form of feelings.

Adaptive agents, which are evolved / programmed to exhibit exploratory behaviors with a curiosity drive as a primary motivator, will likely thrive in their dynamically changing environment. Curiosity as a motivator and exploratory behaviors, which are generally at lower priority (than say avoiding predator), provide default action controlling by way of trial and error (random) or by more strategic interaction with the world. Curiosity and exploratory behavior allow creation of examples and their generalization – acquisition of new schemes to enhance the agents' procedural memory (scheme net).

4.2.2 Behavior Streams

We discussed that the problem of action selection is the decision problem of which action to take at each moment in time, and under observed external and internal stimuli of the moment. The action selection problem can present smaller sub-problems.

Our ASM is a network of behaviors that are partitioned into connected components called behavior streams. Each behavior stream has one or more behaviors and goals, which are dedicated to dealing with a particular sub-problem. A behavior stream is a computational structure in the behavior net that can produce a relevant stream of actions to satisfy a goal. One can also look at a behavior stream as a partial action-plan that controls an agent's actions (internal or external). Figure 4-1 gives an example of a behavior stream or action plan that can compose and send an e-mail acknowledgement message. It also shows how situations and internal motivations influence the behavior stream. Instantiated behavior codelets (discussed in section 4.3.1) in the sideline (discussed in section 4.4.2) spread environmental activation, and drives spread goal-directing activation.

A behavior stream is constructed as a directed graph, which has goal(s) and behavior(s) as nodes and various links connecting the nodes, along which activation spreads as prescribed for a Maes' behavior network. Activation originates from drives, from the situation in the external world, and from internal states. Drives are equivalent to Maes' global goals in the sense that they inject the goal-directing activation energy. A behavior stream can be configured differently (Figure 4-2). In general a behavior stream can be configured: (a) with one or more goal nodes, (b) with one or more behavior nodes, or (c) with one or more behavior and goal nodes. Next, we will discuss the high-level components of a behavior stream: behaviors and goals.

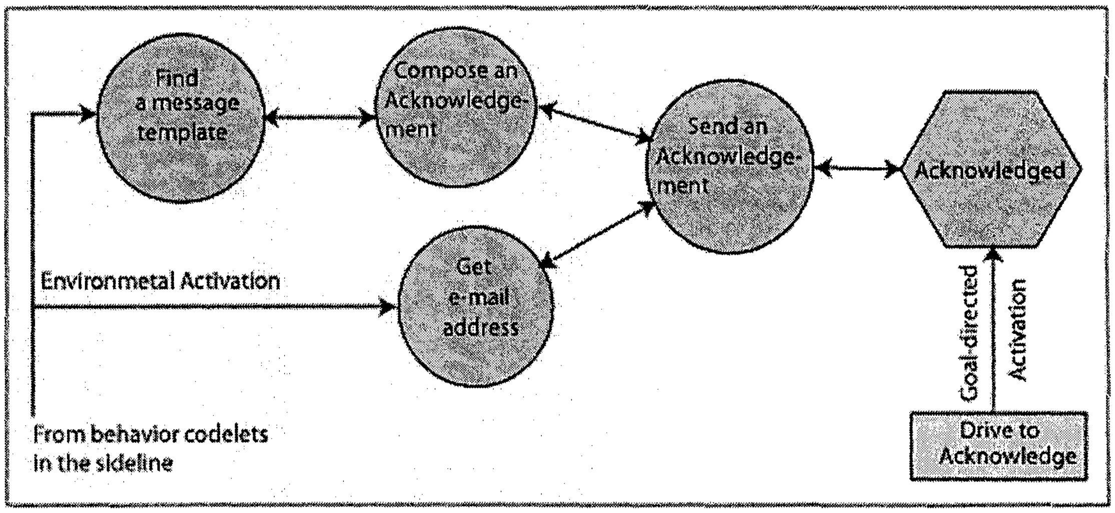

Figure 4-1: An example stream (partial order plan), which can send an acknowledgement e-mail message.

4.2.3 Behaviors

A behavior is our implementation of a goal context as specified in global workspace theory (Baars 1988, 1997). A behavior (Figure 4-3a) is comprised of a *precondition-list*, *variable slots*, *activation*, an *action*, an *add-list* and a *delete-list*. A precondition-list, which is a set of propositions, determines the relevance of the behavior to the current environmental and internal state. The variable slots are placeholders for specific concepts and/or objects that need to be bound for behavioral actions to take place. The variables are bound during the instantiation process, which we will discuss later. Each list has a set of propositions, which will have a true or false value.

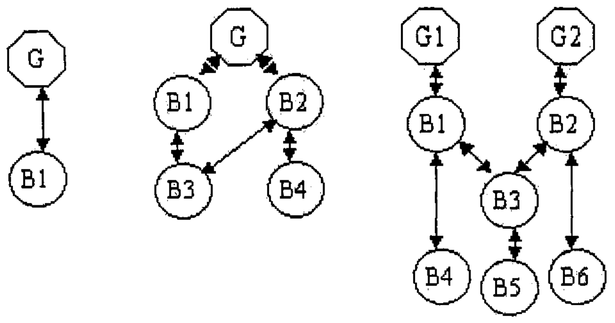

Figure 4-2: Examples of possible stream configurations. (Negatu & Franklin, 2002)

As shown in Figure 4-3b, a behavior constructs each of its different directed links to others' components (behaviors and goals) based on two factors: (a) each link is associated with a shared proposition, (b) the link type (successor, predecessor, conflicter) is determined by the location of the shared proposition in the lists (precondition, add, delete) of the behavior and destination component. Once a behavior is instantiated, it becomes part of the network dynamics of activation/inhibition spreading and the action selection process happens as discussed in section 3.2. That is, when all its propositions in the precondition list are satisfied, an instantiated behavior is *executable.* The activation is accumulated by the behavior in the dynamic process of activation spreading within the behavior net. When a behavior is executable and its activation level is above a threshold value, then it competes globally for execution with other similarly qualifying behaviors. The winner, having the highest *activation*, executes the agent's next *action.* When a behavior is executed, the expected outcome is set (satisfy propositions in add-list and unsatisfy the propositions in delete list), and its activation-level is reset to zero.

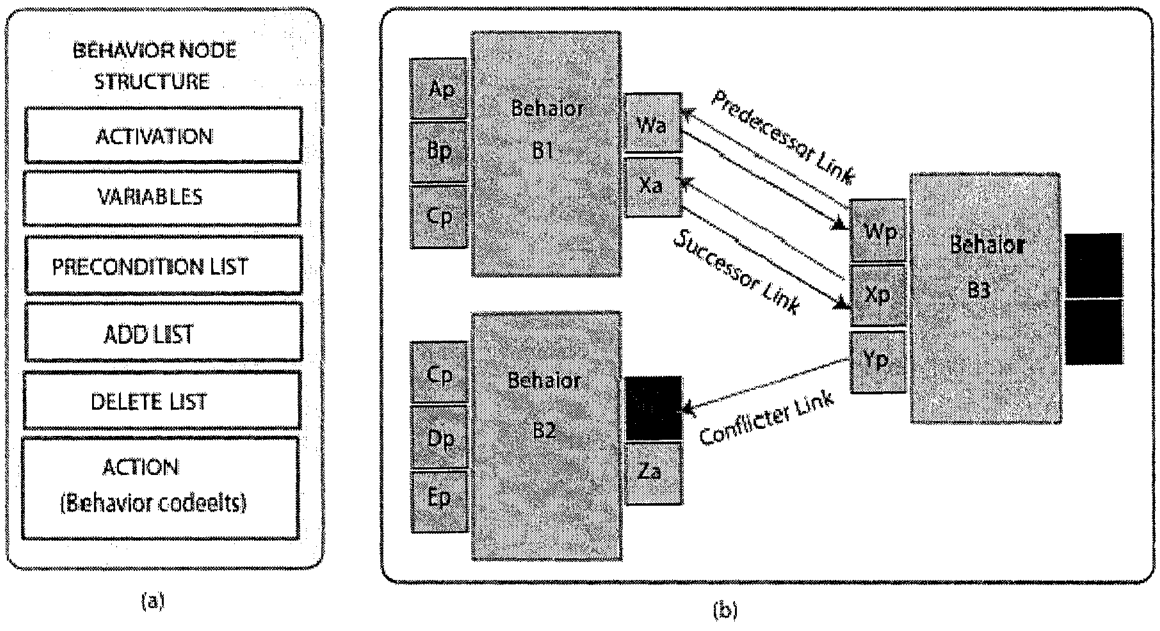

Figure 4-3: (a) Behavior node structure. (b) Building of different links among behavior nodes where the subscripts p, a, and d of a proposition P at behavior Bi correspond to the precondition, addition and deletion lists of Bi in which P is in.; for instance, Y_d at behavior B2 denotes a proposition Y in the delete list of B2.

In the IDA architecture every action is performed by codelets that underlie all higher level constructs. Such codelets, corresponding to some of the processors postulated by global workspace theory, are small pieces of code that execute a small single job. As we go, we will discuss about a number of codelet types associated with a class of functions and programmed to perform specific tasks in each class of function. This hypothesizes the “multiplicity of mind” without uniformity. As a high level construct, each behavior has a number of codelets of different type associated with it; some are for action - behavior codelets, and others are to monitor and evaluate expectation - expectation codelets. We will discuss these codelets in section 4.3 below.

4.2.4 Goals

A goal node is defined as a behavior node without an action. A goal node is satisfyable when all the propositions in its precondition-list are true. A satisfyable goal node infers that all the propositions in its add-list are satisfied and those in its delete-list are unsatisfied. Goal nodes do not have behavior codelets to execute. They simply state when a particular state of affairs is desired and/or achieved.

Without the dynamics of activation spreading, behaviors and goals are simply rules. A behavior is a rule operator that tries to act when its precondition is satisfied, and its action, executed by its behavior codelets, may produce the expected effect specified by its post conditions. Similarly, a goal is an inference rule that declares the achievement of its post conditions when its preconditions are satisfied. Therefore, we can say that a goal node has an implicit action of inferring: stated goal (specified in the add and delete lists) is achieved when all its subgoals (specified in the precondition list) are satisfied. Also, we can think of a goal node defining propositions in its effect as a satisfaction of its preconditions; the abstract level or conceptual depth of the effects or inferred propositions being higher than that of the preconditions.

4.2.5 Variables

Our ASM introduces variables into Maes' network. But, how are variables represented? In our implementation, variables have types that are defined as minimal form of a frame data structure. Each basic variable type will be defined with a "type name" and a "buffer type" attributes, and possibly with a constraint for the buffer value. For instance, one variable type could be an attribute with "type name" of "e-mail-address" and "buffer-type" of "string." Then a variable "sender-address" can be defined as type "e-mail-address." When the variable is in use, a sender's email address (which needs to be a buffer type of "string") will be bound. Variables can be defined at behaviors, goals, and codelets. The variable instance may also have context, which determines the context or problem space under which the variable is being used.

In addition, wherever used, each proposition in the precondition, add, and delete lists could be defined with a variable. The variable, with a defined type, can be bound to a value, which is associated with the assertion of a proposition. For instance, the behavior "send-an-acknowledgement" has a proposition "e-mail-address-ready" in its precondition, and the associated variable of type "receiver-e-mail-address" can be bound to a specific address value, say `john-doe@navy.mil`.

4.3 Behavior Network Module: A Hierarchical Goal Context

According to global workspace theory a goal context (or an intention) is a non-qualitative mental representation about one's own future actions that can dominate central limited capacity or consciousness; it constrains conscious goal images without itself being conscious. Goal contexts perform the following major functions. (1) They help to evoke, shape and produce actions. (2) They can be part of hierarchical structures that allow the agent to deal with tasks that extend over more than one conscious event. (3) They represent future states of the agent, serving to constrain the processes that can help reach those future states. (4) They can be part of hierarchical structures that can constrain a stream of "consciousness." In this section we discuss the high-level implementation of IDA's behavior net module that comprises its goal context hierarchy.

4.3.1 Goal Contexts and Hierarchical Goal Contexts

The behavior network, which has behavior streams as its components, constitutes the goal context system of the "conscious" agent. A behavior in a behavior stream represents a goal context. When this stream gets instantiated, on or more of its behaviors may eventually get executed. If and when a behavior executes, it becomes the *dominant goal context*.

A behavior stream gets its activation from one or more drives and/or other competencies in other behavior streams, as well as from the internal and/or external environment. That is, a stream gets activation from the drives that can be satisfied by its goals or behaviors. A behavior stream also becomes a source of activation to other behavior streams whose goals can satisfy one or more of the preconditions of its behavior(s). This means that there is a goal hierarchy where behavior streams are nested under other behavior streams.

Figure 4-4 illustrates an example of such a goal context hierarchy. Drives are at the top of the hierarchy, where behavior streams serve one or more of these drives directly or indirectly. Behavior stream 1 satisfies the drive directly, while streams 2 and 3 satisfy it indirectly through behavior stream 1. That is, behavior stream 1 receives activation from drive D while behavior streams 2 and 3 get their top down activation from the drive through behavior stream 1. Such structures are not explicitly programmed; they are constructed at run time in an instantiated behavior network system.

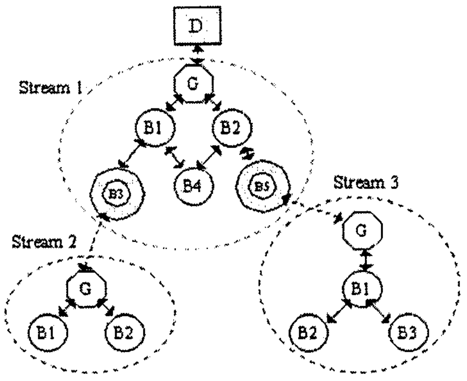

Figure 4-4: An example of a goal context hierarchy that has multiple streams working together. Stream 1 handles its problem only after streams 2 and 3 solve its sub-problems (Negatu & Franklin, 2002).

Behavior B3 has proposition(s) or sub-problems in its precondition list that can be satisfied or solved by behavior stream 2. So, B2 of behavior stream 1 spreads activation to the goal node of behavior stream 2. In the same way, behavior B5 of behavior stream 1 relates to behavior stream 3. The goal of satisfying the drive was carried out in behavior stream 1, which in turn creates sub-goals that are satisfied by behavior stream 2 and behavior stream 3. The creation of goal hierarchies is a run time process. In the process of satisfying the goal of behavior stream 1, new sub-goals are created. The new sub-goals cause the instantiation of behavior stream 2 and behavior stream 3, which have the sub-goals as their goals. Such construction of goal hierarchies with multiple behavior streams is a deliberative process; it takes place with the service of "consciousness" and over multiple cognitive cycles.

Such hierarchical goal structures comprise the goal contexts hierarchies of the "conscious" system. Among many competing goal context hierarchies, the one that has an executing behavior (dominant goal context) is the *dominant goal context hierarchy.*

The style of organization of smaller behavior streams into larger goal hierarchies (Figure 4-4) provides an effective sub-goaling mechanism for our action selection system. The sub-goaling mechanism brings a reasoning capability in the decision making or action selection process. Small behavior streams, which are well defined partial order plans, are solutions to simple routine tasks. Such small behavior streams can be brought together to handle larger routine and non-routine problems. The sub-goaling process is the one that plays the role of 'bringing together' (or the construction of goal hierarchy). In chapter 6, we will discuss how a creation of goal hierarchies provides our agent with the ability to handle impasses, which arise during routine and non-routine problem situations. The sub-goaling allows a goal-directed problem solving. The deliberative sub-goaling process, incorporated in our action selection mechanism, is one of the contributions of this research work to the IDA conceptual model. But the behavior network system also integrates situation directed problem solving; that is, behavior streams relevant to the current environmental situation will be instantiated to possibly be part of goal hierarchies that handle large problems.

The behavior and goal nodes in the behavior network system are processes that define the state of affairs relevant to them in their precondition lists and have the methods to use the precondition state to affect the state of affairs in their add and delete lists. The proposition lists are in conjunctive form and state the propositions with AND logical function. But goal hierarchies can have disjunctive (OR) logical connection. For instance, when two or more behavior or goal nodes have the same proposition in their add lists, then the nodes create an OR connection at nodes that have that proposition in their precondition list.

This is a goal hierarchy that operates in a problem space, and it can be an AND/OR hierarchy. An AND hierarchy, at a precondition list of a node, is successful (executable) when all propositions (sub-goals) in the list are successful. And, an OR hierarchy at a proposition (sub-goal) of a precondition list of a node, is successful if any one of the connected nodes are successful in achieving the sub-goal. So, behavior streams in a goal hierarchy do compete and cooperate to each other to contribute towards handling a problem.

4.3.2 Behavior Codelets

Behavior codelets are the low-level processing units that are related to the behavior net system. Each behavior, being a goal context, is a collection of processes; these processes are its behavior codelets. Behavior codelets function in two different modes: priming mode and action mode. Behavior codelets in priming mode are used to represent internal and/or environment situations and their associated influence on behaviors; the behavior codelets in action mode are used to execute the intended action of their associated executing behaviors.

Behavior codelets in priming mode perform several functions. They listen to "conscious" broadcasts and, when appropriate, respond. They recognize what information is relevant to them, and know what behavior streams to instantiate in the behavior network as a consequence. When a behavior codelet finds itself relevant to a particular broadcast, it spawns a copy of itself with its variables bound to some of the information from the broadcast. The bound codelet jumps to the sideline (see the next section). Once there, the behavior codelet in priming mode has three functions. (1) If necessary, it instantiates behavior streams when the information it carries is relevant to them. (2) It binds information to the variables in behaviors (and their behavior codelets) of appropriate behavior streams. (3) It spreads environmental activation to all the relevant behaviors. In the conceptual model of IDA, these behavior codelets also carry information about feelings and emotions, and provide motivational activation. Behavior codelets in priming mode could instantiate behavior streams for bottom-up or top-down processing in the behavior net dynamics and over multiple cognitive cycles. Their relevance to situational broadcasts binds their preconditions and cause bottom-up processing and their relevance to intentional broadcasts bind their effects and causes top-down processing. The later type behavior codelet priming allows subgoaling and backward reasoning as in the case of non-routine problem solving, which we will discuss in chapter six.

A behavior codelet in action mode has the function of executing its built-in task when its corresponding instantiated behavior executes. When a behavior is instantiated, its variables, if possible, are bound and its behavior codelets reside in the sideline. A behavior executes after being a winner in the dynamics of the action selection mechanism in the behavior network. The executing behavior performs its action via its behavior codelets in their action mode. During execution, behavior codelets use the information in the bound variables.

In a stream priming process, a behavior codelet will instantiate a new instance of a particular behavior stream only if another instance of that same behavior stream in the same operational context is not already instantiated.

Behavior codelets represent fine-grained atomic actions. They encode procedural knowledge and could be defined with their own precondition lists, add lists, and delete lists, where each list is a subset of the corresponding list at the parent behavior. A behavior can have multiple behavior codelets, but with mutually non-interference or without unintended interference to each other. That is, when a behavior is active, its behavior codelets should execute with internal consistency and one's action should not have unintended interference to that of any other. Behavior and behavior codelets are both encode procedural knowledge. In the simplest form, a behavior could have a single underlying behavior codelet and it encodes a much procedural knowledge as that of its behavior codelet.

4.3.3 Evaluating Behavioral Actions

For the control of behavior, we need advance knowledge of expected effects of its actions in the environment. The add and delete lists represent the expected effects or outcome of the completed execution of a behavior. In most behavior net implementations, at the end of its execution, a behavior promises to set the propositions in it's add and delete lists to the expected value, and the system trusts that the behavior kept its promise. This is simply based on the assumption that the behavior codelet(s) under the behavior perform the intended action(s) successfully all the time. This is an unachievable assumption for agent systems that operate in the real world. Therefore, it is necessary to build an independent action evaluation capacity into the mechanism.

Behavioral action, as it is executed by each behavior codelet under a behavior, produces changes in the internal state of an agent or in the external environment, or both. To obtain a proper feedback on the outcome of the behavioral action, an evaluation mechanism needs to have the following capabilities.

- To be able to monitor the events that happen as a result of the action of the behavior and to observe what changes take place internally and/or in the external environment.
- To be able to characterize the changes and evaluate the action by detecting the actual outcome of the action.
- To be able to set the values of the propositions in the add and delete lists of the executing behavior with the detected outcome of the action.
- If necessary, to be able to provide feedback whether the action(s) of the executing behavior has failed by comparing the expected outcome with that of the actual outcome.

Our mechanism implements the action evaluation task by expectation codelets that operate under each behavior. Below we will discuss expectation codelets in more detail.

4.3.4 Expectation Codelets

Expectation codelets, as anticipatory behavior controllers, are important additions in the implementation of the behavior net mechanism. We will discuss facts about expectation codelets; that is, we will highlight their anticipatory controlling functions of monitoring behavioral effects, evaluating of outcomes and reporting possible failures during the execution of a behavior.

Each behavior has at least one expectation codelet under it. When a behavior starts its execution and dispatches its behavior codelets for action, at the same time it activates its expectation codelet(s) so that the real effect or outcome of the action is watched, evaluated/compared and, perhaps, reported. The function of watching the events that are associated with the action of its behavior makes the expectation codelet a particular kind of an attention codelet. The monitored changes as a result of a behavior's action are characterized and formulated as an outcome. The outcome reporting function sets the values of propositions in the add and delete lists, as well as flags related to the reporting of possible failure. The actual outcome reported by an expectation codelet is feedback on the action of the behavior. If an actual outcome of a behavior's action is the same as its expected outcome, then the feedback is positive. If there exists any mismatch of propositional values between the actual and expected outcomes, then the feedback is negative. That is, active expectation codelets always attempt to provide feedback. The feedback is based on the comparison of actual internal sensory consequences (including proprioception, tactile, etc.) and sensed environmental effects of behavioral action with the desired effects. The feedback compilation process may take one or more cognitive cycles. If the action affects only the internal state of an agent, the feedback can be completed in one cognitive cycle. But if an action makes a change in the environment, the feedback may need multiple cognitive cycles. An expectation codelet needs knowledge to carry out its stated functions. Some pieces of required knowledge are built-in during design time and/or more could be acquired via learning. When expectation fails, an expectation codelet tries to bring the detected failure to "consciousness." The expectation mechanism is one of the important additions over Maes' behavior network and one of the major contributions of this dissertation.

The behavior execution, using behavior and expectation codelets, has a functional equivalence to the reafference (Freeman, 1995) process in the brain – when a goal context is activated to produce actions (in the motor system), simultaneously, patterns of expected or predicted changes are passed (to the sensory input) to evaluate performance of the intended action.

As we will see later in chapter 5, if an executed behavior is part of an automated behavioral sequence or automatized procedural skill and if there is a failure (an expected outcome is not fully achieved), then there may be a need to interrupt the automatization process temporarily and bring the involvement of "consciousness." That means, the actions in the sequence of behaviors behind the point of failure are done "consciously". The expectation codelet, as part of the self-organizing automatization mechanism, acts to initiate the process that temporarily disables the automatization effects. As we will also see in chapter 6, a failure of expectation may bring a non-routine problem situation that may bring the problem situation to "consciousness," which, in turn, will recruit the necessary resources to resolve the problem.

4.3.5 Behavior Stream Instantiation

To facilitate our discussion of instantiation, let us first define the following terms.

- *Behavior template or behavior schema:* an instantiable, inactive behavior object with none of its variables bound and none of its associated codelets started.

- *Goal template or goal schema*: an instantiable, inactive goal object with none of its variables bound and none of its associated codelets started.

- *Behavior stream template or behavior stream schema*: an instantiable behavior stream that has behavior and goal templates as its components.

- *Behavior net template or behavior net schema:* a long term procedural memory that stores templates of behaviors, goals and behavior streams.

Any architecture, as the behavior network mechanism is, contains invariants, which include the behavior net template and its contents, the different codelets that underlie its behaviors, the many global parameters, the fashion of linking nodes in the network, the algorithm that governs the activation/inhibition dynamics, and some others. The behavior or nature of the network is the product of the invariant architecture and the content. The content binds values or objects to the architecture. When a behavior network system is initialized, every part of the architecture is ready, but we will not see its action until a content is bound. The first binding is the starting of the behavior codelets in priming mode and drives. The behavior codelets allow the behavior network to be coupled to its environment and its internal state. The drives set the primary motivation. There will not be spreading dynamics in the network until behaviors and goals are bound with content. Unbound behavior codelets are the base or primitive or atomic procedural templates or schemes. The behavior and goal templates represent a procedural knowledge with higher level of abstraction than behavior codelets. Behavior streams templates are more complex procedural schemes that encompass behavior and goal templates as connected components. The behavior net template constitutes the complete long-term procedural memory of an agent. All templates with high-level abstraction are definable as a coalition of one or more behavior codelet templates.

For a given sub-problem in response to the environment, one or more behavior streams can be relevant. When a behavior codelet, in priming mode and in response to the environment, considers a behavior stream relevant, the instantiation mechanism creates an instance of the stream from its template that includes the binding of variables in its behavior(s) and goal(s) wherever possible. That is, the instantiation is a process that searches a relevant behavior stream template, binds the behavior stream with the required objects, and allows the behavior stream to be active in the network dynamics. We will discuss the details of the computational mechanisms of the instantiation process in section 4.4.

4.3.6 Discrimination-Factor Parameter

Instantiated behavior streams could directly interact in the dynamics of the behavior net. Interactions happen if there exists a behavior in one behavior stream that passes activation energy to a behavior in another behavior stream. Such interactions allow having a net lose of motivation energy from one behavior stream to another. Over time, a behavior stream with weak relevance to environmental or intentional situation could become active due to such intra behavior stream activation passing. We introduce a global parameter that we called *discrimination-factor* (ξ) to control such activation passing. This parameter is to have a real value in (0, 1]. A value of zero means discrimination of intra behavior stream activation passing is disabled and the dynamics in this aspect is similar to that of Maes' algorithm. But, a value of one implies that full blocking of the intra behavior stream activation passing; that means, the average activation of behavior streams is determined by the amount of input they get directly from external activation sources. With high discrimination-factor value, behavior streams with high intentional and environmental relevance are likely to become dominant over less relevant behavior streams. Later we will see how this parameter along with the importance parameter values associated with drives will allow controlling of priority of competing behavior streams or goal context hierarchies. Such priority control is an important addition over Maes' mechanism and it is one of our significant research contributions. Maes' mechanism could provide control of such priorities only by built-in rules. As we will discuss later, rule based control of priorities has disadvantage compared to parametric control of priorities particularly if the agent environment is dynamic.

4.3.7 Disposing of Action Plans

If an instantiated stream satisfies a drive/sub-goal, then this stream terminates itself as soon as it completes its actions or its effects are satisfied either by the actions of other behaviors or external actors. The associated codelets die soon afterwards. In a dynamic domain, it is possible that an instantiated stream may never get a chance to finish its actions. There needs to be a way to forget about such streams after some time. In IDA, streams (with their goals and behaviors), behavior codelets and expectation codelets are designed with a decay mechanism on their livelihood. As time passes without activity, their livelihood strength decays; and all die when their strength levels fall below a threshold. We have added one more global parameter and this parameter determines the *base decay rate* (ε) in the behavior network.

4.4 Working with "Consciousness"

If the behavior network constitutes the active goal contexts of a "conscious" system, then it should be integrated to the "consciousness" module in order to effectively influence "conscious" activities and to play its action selection role.

Typically, an action performed by a dominant goal context leads to a conscious experience, and this conscious experience in turn could evoke a goal context whose action will influence the next conscious experience. To adhere to global workspace theory, the behavior network system and the "consciousness" mechanism should be coupled in such a way that a conscious event (via its broadcast of information content) influences which behavior (goal context) becomes active (dominant) in the immediate future. In turn, the execution of this dominant goal context should constrain what comes to "consciousness" next. Figure 4-4 will help explain how the action selection and "consciousness" modules are coupled and work together.

As per global workspace theory, a goal context is a coalition of processors. In this case the behavior codelets are identified with the behavior they underlie, but at a different level of abstraction. Behaviors and behavior codelets bring different levels of dynamics to the action selection process. Here's how it works.

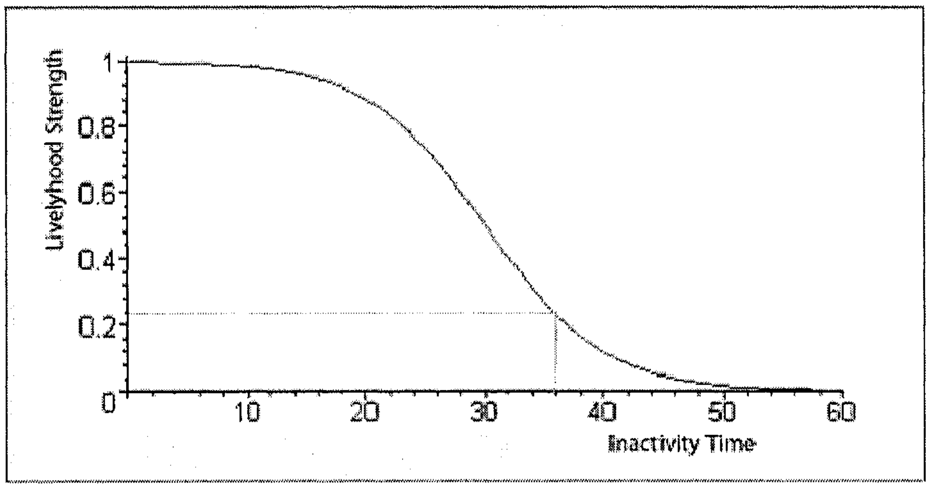

Figure 4-5: Base decay curve used to decay behavior streams and codelets. A behavior stream is uninstantiated if its livelihood strength is below a threshold. The threshold of 0.24 livelihood-strength is reached when the inactivity stretches over 36 time units. The curve and the threshold can be adjusted to the domain requirement.

4.4.1 Skybox

The skybox is an abstract place where instantiated streams reside. Instantiated streams form the active behavior network, whose dynamics are implemented by mechanisms in the skybox. The dynamics and characteristics of the active behavior network in the skybox are the same as that of the Maes' behavior network system.

One or more behavior streams can be instantiated at a time. If these streams are operating in the same context or problem space, then the behavior streams could pass activation or inhibition to each other. This happens if there is a proposition (whose variable is bound to a value associated with the context of the problem space and is a state item shared by the behavior streams) that could allow a construction of a valid link that connects a competence module of one behavior stream to a competence module of another behavior stream. Behavior streams, which operate in the same problem space, share the same activation dynamics plain and they can be considered as being merged together as a single active behavior stream. If any two behavior streams are instantiated to operate on different problem spaces, then they are in the skybox at different levels – no activation/inhibition spreading link that connects the behavior streams. So, behavior streams at different levels do not share the activation dynamics plain. However, their behaviors compete globally in order to become the next dominant goal context. We like to note that a behavior stream template could be instantiated to multiple instances - each instance to be bound to a different context or to operate on different problem space. *BNManager* is the component module that implements the functions of the skybox.

4.4.2 Sideline

The sideline is an abstract level where instantiated behavior codelets reside. Behavior codelets with priming-mode bind variables to their behaviors in the skybox. Behavior codelets with action-mode get their binding information and activation-level setting from their instantiated behaviors. That is, an instance of a behavior codelet with priming-mode binds its information content to variables of a behavior it primes and instantiated in the skybox; in turn, the behavior binds the relevant information to variables in its underlying action-mode instances of behavior codelets in the sideline. In addition, the sideline has other functions.

1. Behavior codelets perform their tasks of priming, binding and sending activation to their relevant behaviors.

2. Heuristic rules are kept that enhance the accuracy of the priming decisions. One simple heuristic is instantiate all behavior streams that are partially relevant. This simplicity comes at the cost of computational overhead - allows instantiation of many unexecutable behavior streams that eventually decay and terminate.

3. Bound behavior codelets form coalitions and jump to the playing field when their corresponding behaviors in the skybox become active.

4. Behavior codelets return to the sideline when they finish their tasks in the playing field. Behavior codelets can't just die after they finish their execution and this requirement will be clarified when we discuss about automatization in chapter 5.

The *Sideline Manager* component of our action selection mechanism implements the sideline functions.

4.4.3 How All Work Together

Figures 4-6 and 4-7 illustrate how the behavior net module and "consciousness" work together. IDA starts, with all its mechanisms and built-in knowledge distributed over its modules. For instance, the action selection mechanism has built-in domain knowledge in the behavior net specification and in all the implemented behavior codelets.

Attention codelets are watchful for some particular occurrences that might call for "conscious" intervention. Mostly, attention codelets are watching situations reflected in the working memory, which will likely contain contextualized perceptual content and intermediate internal information. *Information codelets* carry the information describing a situation. Upon encountering a situation, the appropriate attention codelet will form a coalition with the small number of information codelets. *Attention codelets* collect associated *information codelets* containing information about internal or external states of the agent and compete for "consciousness." The "consciousness" mechanism picks the winning coalition of attention and associated information codelets, and broadcasts their content in order to recruit internal resources that help to produce appropriate actions.

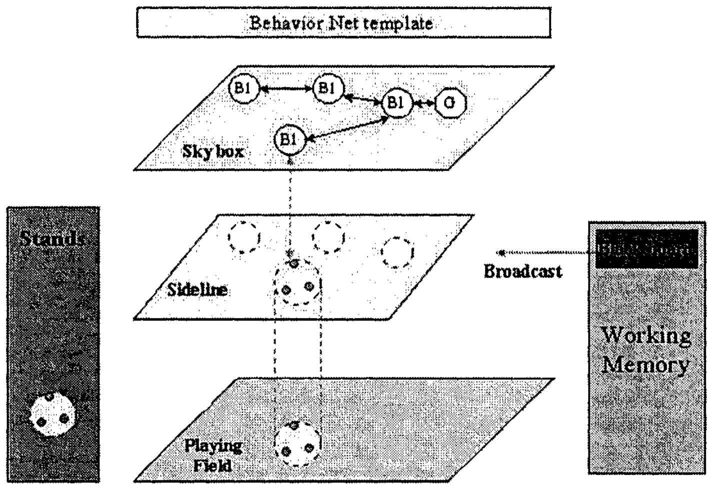

Figure 4-6: A behavior in action with its involved components of "consciousness" and the ASM module. Behavior codelets in the stands receive a global broadcast of conscious content (from the blackboard) and prime a relevant stream; the stream gets instantiated in the skybox from its template. When the dynamics in the skybox selects a behavior, then its corresponding coalition of codelets acts after jumping from the sideline to the playing field (Negatu & Franklin, 2002).

Suppose, for example, IDA receives a message from a sailor saying that his projected rotation date (PRD) is approaching and asking that a job be found for him. The perception module would recognize the sailor's name and social security number (ssn), and that the message is of please-find-job type. This information would then be written to the working memory. The general principle here is that the contents of perception are written to working memory before becoming conscious. The storing of the contextual detail of the percepts in the working memory is the result of the first three steps in IDA's cognitive cycle: (1) preconscious perception to create meaningful percepts from sensed internal and external stimuli, (2) the storage of percepts in the preconscious buffer of the working memory, and (3) the automatic retrieval of local associations from long-term associative memory and transient episodic memory using the content of the preconscious buffer.

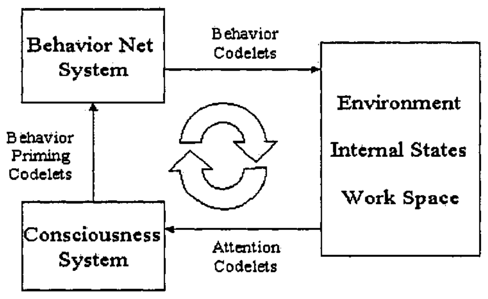

Figure 4-7: Interleaving of unconscious goal contexts and their actions with "conscious" events. Attention codelets brings information to "consciousness," "consciousness" broadcasts conscious content, behavior codelets receive broadcast and prime relevant behavior streams, and instantiated streams act to affect changes. (Negatu & Franklin, 2002).

An attention codelet will note the please-find-job message type, gather information codelets carrying name, ssn, and message type, which will be formed into a coalition, and compete for "consciousness." The competition for "consciousness" is the 4th step of IDA's cognitive cycle (or CC-4). If or when successful, its content will be broadcast (CC-5). Among behavior codelets waiting in the stands, some find themselves relevant to the broadcast, make a copy of themselves with variables bound with information from the broadcast and jump to the sideline; that is, the "conscious" broadcast recruits the relevant resources (CC-6). Once there, they collectively or individually prime all the relevant behavior streams. The *BNManager* retrieves all the primed behavior streams from the *behavior net template* memory and make them available for instantiation. Instantiated behavior streams become part of the dynamics of the behavior net. Then the behavior codelets bind behavioral variables, assert relevant propositions, and send environmental activation to the appropriate behaviors of the instantiated behavior streams. At the same time, corresponding to each instantiated behavior, a coalition of behavior codelets, with the appropriate information bound, is instantiated in the sideline and waits for execution. The just instantiated behavior streams constitute the goal context hierarchies (CC-7) relevant to the broadcast. If a behavior codelet has an instance in the side line and it primes an instantiated behavior stream, a new instantiation will not happen. It simply binds the current information to the old instance.

The dynamics of the behavior net eventually picks the dominant goal context or behavior for action (CC-8). The corresponding coalition of processes (the behavior codelets) waiting in the sideline jumps to the playing field and performs the designed action (CC-9), using the bound information. As part of the action and along with the behavior codelets, the sibling expectation codelets also jump to the playing field to monitor and evaluate the action, and report the outcome. After finishing their task, behavior and expectation codelets return to the sideline and wait there until they are activated for another cycle of action or they decay away. The decay happens as time passes without activity and when their activation levels have decayed below a specified threshold value. The corresponding behavior that resides in the skybox may be uninstantiated while its codelets stay around until they decay away.

Returning to the specifics of our example, the broadcast is received by appropriate behavior codelets, which will instantiate a behavior stream for reading the sailor's personal record. They also bind variables with name and ssn, assert propositions that can satisfy the precondition of the behavior, and spread environmental activation. The behavior that knows how to access the database can be chosen by the dynamics in the behavior net and can be executed. When the behavior executes, its behavior codelets read data from the sailor's file and write the data to the workspace. Each such write results in another round of a cognitive cycle: perception, associations, the triggering of attention codelets, competition among them for "consciousness," coming to "consciousness" or broadcasting of the content of the winner attention codelets, the recruitment of behavior codelets, instantiation of behavior streams, the selection of next behavior, and the action of behavior codelets along with the monitoring, evaluation, and reporting role of expectation codelets. As long as it's the most important activity going, this process continues until all the relevant personal data is written to the workspace. Such repeated runs through "consciousness" and the behavior net result in course selection of possible suitable jobs being made from the job requisition database. The dynamics in the behavior net may be influenced by emotional "experiences," such as guilt at not getting a sailor's orders out on time and frustration at not understanding a message.

IDA has a behavior stream for a constraint satisfaction system designed around a linear functional. It provides a numerical measure of the suitability of a specific job for a given sailor. Here issues such as moving costs, Navy policy, etc. are considered. With the run of such a behavior stream, the job's fitness value is written to the workspace.

IDA's domain is fairly complex and it requires *deliberation* in the sense of creating possible scenarios, partial plan of actions, and choosing between them (Sloman 1999). In our example, IDA now has a list of a number of possible jobs in her workspace, together with their fitness values. It must construct a temporal scenario for at least a few of these possible billets to see if the timing will work out (say if the sailor can be aboard ship before a departure date) (Franklin, 2000). In each scenario, the sailor leaves his/her current post during a certain time interval, spends a specific length of time on leave, possibly reports to a training facility on a certain date, uses travel time, and arrives at the new billet with a given time frame. Such scenarios are valued on how well they fit the temporal constraints (the gap) and on moving and training costs. These scenarios are composed of scenes organized around events and are constructed in the workspace by runs in the behavior net and "consciousness" modules as discussed before.

We humans most often select actions subconsciously. That is, without conscious thought. But we also make voluntary choices of action, often as a result of the kind of deliberation described above. Baars argues that such a voluntary choice is the same as a conscious choice (1997, p. 131). We must carefully distinguish between being conscious of the results of an action and consciously deciding to take that action, which is, consciously deliberating on the action. It is the later case that constitutes *voluntary action*. William James proposed the ideomotor theory of voluntary action (James, 1890). James suggests that any idea (internal proposal) for an action that comes to mind (to "consciousness") is acted upon unless it provokes some opposing idea or some counter proposal. Global workspace theory adopts James' ideomotor theory as is (Baars, 1988), and provides a functional architecture for it. The IDA model furnishes an underlying mechanism that implements the theory of volition and its architecture in a software agent (Franklin, 2000; Kondadadi & Franklin, 2001).

Table 4-1: Design assumptions and hypothesis related to action selection mechanism of IDA (Negatu & Franklin, 2002).

ASM Component	Design Assumptions or Hypotheses in the ASM System
Motivation	The hierarchy of goal contexts is fueled at the top by drives, the prime motivators, and at the bottom by input from the environment, both internal and external.
Significance of Action	An action attracts involvement of "consciousness" in proportion to the motivation level of the dominant goal context that perform the action.
Voluntary Action	A timekeeper, who becomes less patient as the time for a decision increases, controls voluntary action in humans. Each time a proposal or objection reaches "consciousness," its chance of becoming conscious again diminishes.
Avoidance of Goal Conflict	All goal conflicts are not informative and do not need conscious intervention. Goal conflicts can be avoided unconsciously in the instantiated hierarchical goal context via spreading of inhibitive activation energy.
Choice Points in Goal Hierarchy	Hierarchical Goal Contexts can have choice points, and some decisions at choice points can be made unconsciously mainly based on situational motivation. Choice points in the action control bring uncertainty and

	conscious intervention, but not in all.
Goal Hierarchy Instantiation	Goal hierarchy is instantiated either by a conscious event or unconscious process. The unconscious one happens via a priming effect from a preattentive or subliminal perception process that senses the internal state and/or the environment.
Action Types	Three action types in a hierarchical goal context: consciously mediated, unconscious, and voluntary.
Drives as Self-Concept	Drives and their importance values are part of a self-concept as they are the deeper parts of Goal Context Hierarchies that shape experiences in self-monitoring.

Suppose that in our example at least one scenario has been successfully constructed in the workspace. The players in this decision making process include several proposing attention codelets and a timekeeper codelet. A proposing attention codelet's task is to propose that a certain job be offered to the sailor. That is choosing a job to propose on the basis of the codelet's particular pattern of preference (different issues like priority, moving cost, gap, etc each with weight assigned to it). For example, our proposing attention codelet may place great weight on low moving cost, some weight on job fitness value, and little weights on others. This codelet may propose the second job on the scenario list because of its low cost and high fitness, in spite of low priority and sizable gap. If no other proposing attention codelet objects (by bringing itself to "consciousness" with an objecting message) and no other such codelet proposes a different job within a given span of time, the timekeeper codelet will mark the proposed as being one to be offered. If an objection or a new proposed job is made in a timely fashion, no job will be offered.

Two proposing attention codelets may alternatively propose the same two jobs several times. Several mechanisms tend to prevent continuing oscillation. Each time a codelet proposes the same job, it does so with less activation and, so, has less chance of coming to "consciousness" subsequently. Also, the timekeeper loses patience as the process continues, thereby diminishing the time span required for a decision. Finally, the metacognitive module watches the whole process and intervenes if things get too bad (Zhang et al., 1998a). This mechanism, which uses Jackson's pandemonium theory as the computational model, leads us to the voluntary action hypothesis in Table 4-1. The behavior codelets are implemented with the facility to process voluntary actions.

Back to our example, once one or more jobs are voluntarily selected, IDA must generate an e-mail message offering them to the sailor. Due to its relatively narrow domain, IDA generates language (email messages) by filling in appropriate scripts. The scripts reside in behavior codelets and are composed by *consciously mediated actions* of behavior stream(s). Thus, much of the language production results from filling in the blanks in learned scripts, and concatenating the results.

4.4.4 Differences with Basic Blackboard Model

Here, we try to briefly discuss the difference between IDA's architecture and the basic blackboard model (Nii, 1986). The basic blackboard model has three components: the *blackboard* or global data structure, the *knowledge sources* or KSs, and the *control*. KSs are independent computational modules embodying specialized knowledge that is useful in solving a sub-problem in the context of a global application. KSs are self-enabling and opportunistic and can communicate only though the blackboard. Control directs the overall computation of a system. The basic control cycle (a) determines enabled KSs, (b) chooses KSs for execution among the enabled, (c) executes the chosen KSs, which will cause a change in the blackboard, and (d) repeats the cycle. The functional parallel is that blackboard, KSs, and control respectively correspond to the global workspace, contexts, and global broadcasts (conscious events) in IDA. Functionally, both global broadcast and control are used to exercise global coordination. We will point some of the differences.

1. Unlike KSs, contexts can communicate without global broadcasts.

2. Blackboard is a large working memory while IDA's global workspace has a limited capacity.

3. *Control* picks the KSs that execute, but the global broadcast mostly influences the contexts, except in the case of voluntary control.

4. Control chooses the KSs that access the blackboard while the conscious event chooses the content to be accessed by all.

5. More importantly, the basic blackboard model is a general architecture for computational systems, while IDA is a cognitive agent architecture that implements the global workspace theory, and as such, it's a human-like information-processing system and is mostly constrained by cognitive and/or neurological evidence.

4.5 Issues Related to Action Selection Mechanisms

IDA's architecture in general and its ASM system in particular have been designed to be faithful to global workspace theory. Table 4-1 focuses on the important design assumptions or hypotheses related to the ASM system or the concordance of the ASM to global workspace theory; in this section we briefly discuss some of these.

4.5.1 Types of Behavior Stream of Actions

Actions are unconscious and happen in a response to some external or internal change, and they in turn cause changes in the system. Special processors (attention codelets) bring perceptions and thoughts to "consciousness." Other unconscious processors (behavior codelets), which find relevance to the conscious contents, instantiate appropriate goal context hierarchies. The internal and environmental motivations influence the instantiated behavior streams that unconsciously select the appropriate behavior, and the underlying behavior codelets perform internal and/or external actions. In our mechanism, we can categorize actions into three types based on the relationship of the behavior stream that controls the actions to conscious experiences. That is, how a stream relates to a conscious experience that precedes it and the conscious experience(s) during stream execution. Please note that we differentiate between the conscious experience of the results of actions and the conscious deliberation on taking the action. The later is the case of voluntary action (section 4.4.3).

4.5.1.1 Consciously Mediated Stream of Actions

For a consciously mediated stream of actions, the controlling stream (dominant goal context hierarchy) is triggered by a conscious event via behavior codelets. The triggering conscious event may be a voluntary decision to select the behaviors (goal context) in the stream. It may also be the case that the triggering conscious event could not be explicitly conscious of any of the behaviors in the stream. During the execution of such a stream, one or more of its behavioral actions can cause informative changes to make it to "consciousness" by some attention codelet. That is, the effect of the actions becomes a conscious experience and the information is broadcast. The broadcast information can activate behavior codelets that can reinforce the already instantiated stream and bind and/or rebind some variable slots of the behaviors. More behavioral actions of the stream can cause more conscious experiences. This way, before an instantiated stream, as a goal hierarchy, achieves its stated goal, a number of conscious-unconscious-conscious (CUC) triads could happen. If a stream causes more than one CUC triad during its execution, then the actions of the steam are called *consciously mediated actions*. In general, action-induced conscious experiences can activate behavior codelets that could instantiate new competing/cooperating stream(s).

Consciously mediated behavior streams are well defined partial action plans. Such behavior streams may be activated via a voluntary action selection, but once they are active, they can execute their stream of actions without the involvement of any voluntary decision or conscious goal selection. But they are consciously mediated so as to provide the necessary data or environmental coherence to perform some of their actions. For instance, sitting at my living room, I may voluntarily decide to drink a soda. This entails walking to the refrigerator, opening the refrigerator door, picking up a can of soda, opening the can, and drinking from the can. In this sequence of actions, I will be conscious of some things without involving voluntary decision such as the handle's position on the refrigerator door, a specific location of a can in the refrigerator, the location of the can's opening tag. These conscious contents are highly constrained by the executing sequence of actions (dominant goal hierarchy). As such, they are used to bind information for my action of opening the refrigerator door, my action of opening the can, my action of drinking from the can, and others. Without making my empty hand close enough and aligned with the door handle, I cannot open the door. Consciousness mediates the intermediate actions that are performed under a routine goal context hierarchy.

4.5.1.2 Unconscious Stream of Actions

These types of actions are produced by streams instantiated in the same way as consciously-mediated actions, but none of the actions of the behaviors in the stream trigger a conscious event. As a result, once the stream is instantiated, it completes its execution unconsciously except possibly for the last goal state achieved at the end of the stream execution; i.e., at most, only one CUC triad. Unconscious streams of actions are built-in or learnt via evolution or self experience.

4.5.1.3 Stream of Actions with Voluntary Selected Goal

Here the instantiation of a behavior stream is triggered by a voluntary decision. That is, the goal of the stream is in "consciousness" as a goal-image before the behavior stream is instantiated in the behavior net. In other words, a behavior stream is instantiated by its own goal-image. The actions of the instantiated behavior stream, which satisfy the voluntary selected goal-image, may or may not be consciously mediated.

Through *experience* or *over-learning*, consciously mediated streams of actions can become unconscious streams of action - automatization of stream of actions, which we will discuss in chapter 5.

4.5.2 Characteristics of ASM as a Goal Context Hierarchy

Goal significance: If an instantiated stream has a behavior with a precondition (sub-goal) that has not been satisfied for some period of time, then the stream will be stuck and it cannot satisfy its goal(s). An attention codelet, with the appropriate motivation (activation) level or the significance of the stuck behavior, can try to bring the situation to "consciousness." The broadcast of the situation will allow recruiting of other streams that can resolve the sticking point. This means, the "consciousness" mechanism facilitates the process of building goal hierarchies (like the one in Figure 4-3) dynamically in the active behavior net system. Global workspace theory states that a goal/sub-goal with enough significance is considered to be informative and may trigger a "conscious" intervention, hence, a concordance with the theory.

Problem solving: Conscious mediation is helpful in recruiting a number of streams that cooperate under a goal hierarchy to create the unconscious process of problem solving. Each instantiated stream is an island of a partial solution, and the conscious mediation (with its role for global coordination and control) helps to connect the different islands to reach the complete solution. The problem solving could proceed in both top-down and bottom-up directions. Some islands start to build down from the goal states, and some islands start to build up from the problem initial state. The building continues until a solution to the problem is found or until the problem is found to be unsolvable.

Bias to dominant goal hierarchy: A stream that has an executing behavior is an active action-plan. The behavior net algorithm has an implicit bias that gives momentum for the completion of an active plan over other competing plans that may exist. This is due to the fact that activation level of behaviors does not get reinitialized except for the behavior that was just active. That is, the past activation dynamics creates an activation momentum that biases the selection dynamics towards matured active plans; therefore, it is likely that the next active behavior to be from the active stream. This fits the global workspace theory in which Dominant Goal Hierarchies, consisting of a coherent stream of goal contexts, have a tendency to remain dominant until their goal is achieved. At any moment, but with less likelihood (i.e., deterministic), any goal context from a competing goal context hierarchy can disrupt the dominant goal hierarchy.

Unconscious goal conflict resolution: The behavior net resolves goal conflicts at the goal and behavior nodes by spreading inhibitive activations. This hypothesizes (Table 4-1) that some goal conflicts in goal context hierarchies can be resolved unconsciously in the ASM system; not all goal conflicts are novel.

Unconscious decision of choice points in goal hierarchies: Two or more behaviors or streams can satisfy a sub-goal of another stream. Satisfying this sub-goal creates a choice point; i.e., two chose one of the competing streams. The unconscious process of activation dynamics in the behavior net will make one choice point more relevant than the other one, mainly based on a situational motivation; thus not all choice points in goal hierarchies need conscious intervention.

4.5.3 External Control of Behavior Network

The personality (motivational and behavioral) control of the action selection mechanism can be accomplished by adjusting the factors that influence the dynamics of activation/inhibition, which in turn influence the relative strength of a behavior, the relative bias towards one stream over others, the relative bias of the system to goal-orientedness or situation-orientedness, and other properties. The implementation of the behavior network provides the necessary interface to have external control to the following parameters.

τ - Drive importance parameter, which is local to each drive, determines the relative motivational importance of a drive. This provides a motivational control in the system.

θ - Activation level threshold global parameter above which an executable behavior can be active.

γ - Drive activation energy global parameter that determines the amount of activation energy injected to satisfying streams.

ϕ - Environmental activation energy global parameter that determines the amount of activation energy injected by behavior codelets in priming mode to relevant behaviors and goals.

ξ - Discrimination-factor global parameter that determines the level of spreading of activation energy among interacting behavior streams.

λ - Instantiation threshold global parameter that determines whether a new instantiation is possible or not. This is a memory resource management control. If too many behavior streams are instantiated, the behavior codelets that prime new streams should have an activation level over this threshold.

ε - Base decay rate global parameter determines the rate of decay for instantiated streams and codelets. This parameter is weighted by another local parameter to get the actual decay rate used by streams and codelets.

These and other parameters can be externally controlled automatically with a metacognitive system, which is a module in the conceptual architecture of IDA. All the global parameters are actively monitored and their adjustment will have run-time effect on the system.

Regardless of the motivational influence or environmental effect on the selection of the next action, an external control can force a different action at any time. This is another type of control, and in our system this role is accomplished by voluntary actions, which is discussed in section 4.4.3.

4.6 Implementation Details

We have already discussed that brain is a highly distributed and parallel system. Accordingly, many models, including global workspace theory, hypothesize the 'multiplicity of mind.' The implementation of the action selection system is motivated to be true to this fact, and all the critical computing components run as independent threads. We use the java programming language, and the parallelism uses the multi-thread infrastructure provided by the java virtual machine. That is, critical components (behaviors, goals, propositions, links, drives, etc.) of our behavior network are implemented with a parallel algorithm. Distributed programming brings its own complexities and associated challenges, such as the problems of synchronization and dead-lock avoidance.

The author has devoted great effort and time to the design and implementation of a behavior network system having the following properties.

1. It is a *Domain independent* behavior network system with a simplified facility to integrate domain specific knowledge. One has to simply program the codelets, which are specific to a domain, and has to program an XML formatted behavior specification with a given programming language called BNDL (Behavior Network Description Language). All are needed to have a domain dependent behavior network system.

2. It has general representation and control flow that can accommodate different action selection features, although all are not fully exploited by the IDA domain.

3. It can be isolated from the "consciousness" and other modules of IDA, and it can be used as a standalone independent agent development platform. As such, it can be used for agent domains that are appropriate for behavior based system.

4. Good coding practice is followed; proper and consistent class, variable and method naming conventions are adopted, and each source code file is well commented.

5. Development of domain application is well structured and relatively easy. Besides the facility to specify behavior network design in an XML file, different base codelet classes are implemented, and domain dependent codelets need to override few methods or implement few abstract methods.

Next, we will briefly discuss the implementation of the critical components. Note that our discussion of the algorithms will be limited to the major steps and will not focus on the synchronization issues associated with the parallel programming.

4.6.1 Behavior Network Starter

This is the component (DriveStartup.class) that runs as a thread and starts the behavior network system from a given behavior network specification file and pool of codelet classes that may be located at different directories. It also hooks the behavior network system with other necessary modules such as the long-term working memory. Specifically, the behavior network starter does a one time initialization of the necessary objects of the behavior network and then does its recurrent task.

The initialization steps of the behavior network starter are (refer to Figure 4-8):

1. Compile the XML formatted domain specification of a behavior network to a *behavior network template* that contains the instantiable objects of the network.

The compilation is done by the *BNDL Parser* (BNDLParser.class). The behavior network template contains templates of streams, behaviors, goals, and drives, and as such can be viewed as a *long-term procedural memory*.

2. Setup the global parameters of the behavior network for global accessibility.

3. Initialize the *side-line manager*, whose function is discussed in section 4.4.2.

4. Initialize *codelet list manager*. The codelet list manager stores initialized behavior generator codelets that are relevant to the behavior network. Generator behavior codelets are always in the stands, and they respond to relevant conscious broadcasts. When they become relevant to some broadcast, they spawn an instance of themselves that is bound by the broadcast information. This is another long-term procedural memory system.

5. Initialize *state list manager*. The state list manager stores initialized proposition objects, where propositions are those specified in the precondition, add, and delete lists of behaviors, goals, and drives. The content of the state list manager represents the internal state of the behavior network system.

6. Initialize all proposition objects and store them in state list manager. Proposition information in the behavior network template is used in creating the corresponding objects. Each proposition can be considered as a declarative memory item.

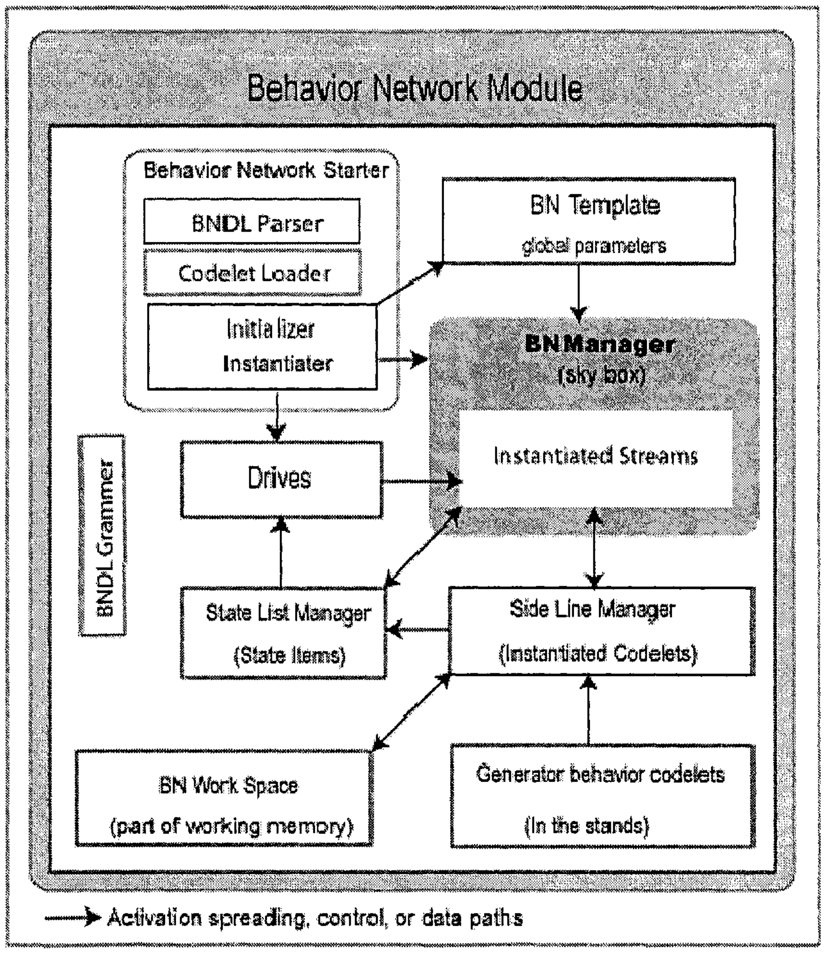

Figure 4-8: Behavior Network module architecture.

7. Initialize generator behavior codelets and store them in the codelet list manager. The codelet information in the behavior network template is used to start the generator behavior codelet daemon. The *codelet loader* (CodeletLoader.class) is used to initialize the codelet objects from codelet classes; it is a critical capability towards the generality and the domain independence of our implementation.

8. Initialize drives. At this point, drives are ready in the dynamics of action selection.

The *behavior network starter* goes through the following loop.

1. Sleep for a short length of time.

2. If no priming event is queued, go to step 1.

3. Using the information in the priming event, retrieve the stream template from the behavior network template.

4. Start the stream thread object with the necessary initialization parameters

4.6.2 Behavior Network Specification

4.6.2.1 BNDL: Behavior Net Definition Language

Part of domain knowledge is specified in an XML formatted file. The different components of the behavior network are defined in the XML file using BNDL (Behavior Network Description Language) grammar. Figure 4-9 partially shows implementation of the BNDL grammar in the document type definition (dtd) format.

The third line (Figure 4-9) defines the '*behaviornet*' element that contains one or more streams and one or more drives. Behavior codelets (bcodelet) can be specified here or under behaviors. The dm-type is an element that contains memory type objects, which are used to define variables. The dm-instance is an element that contains memory instances that are defined at design time or that are needed for initialization. The memory instances should have types defined in the dm-type element. The general-info is an element that contains attributes and elements to specify general information including the title and short description of the domain, the names of the mechanism developer and domain application developer, and version information.

Besides, as shown in lines 4 to 9, the *behaviornet* element has attributes, which define the global parameters used in the behavior net and its element-name. Figure 4-9 also shows the grammar that defines behavior streams and drives. Stream element has an attribute for element-name and contains goal and behavior elements.

```
<?xml version='1.0' encoding='us-ascii'?>
<!-- Behavior Net Description Language Document Type Definition -->
<!ELEMENT behaviornet (stream+, drive+, bcodelet*, dm-type*, dm-instance*, general-info*)>
<!ATTLIST behaviornet
        element-name   NMTOKEN   #REQUIRED
        theta    NMTOKEN   #REQUIRED
        gamma    NMTOKEN   #REQUIRED
        pi      NMTOKEN  #REQUIRED
        delta    NMTOKEN   #REQUIRED
        phi      NMTOKEN   #REQUIRED
        decay-rate NMTOKEN   #IMPLIED
        >

<!ELEMENT stream (behavior+, goal+)>
<!ATTLIST stream
        element-name NMTOKEN #REQUIRED
        >

<!ELEMENT drive (precondition+, alias*, attncodelet*)>
<!ATTLIST drive
        element-name   NMTOKEN #REQUIRED
        importance     NMTOKEN #REQUIRED
        intensity     NMTOKEN #IMPLIED
        satisfaction   (once | continuous | periodic) #REQUIRED
        codelet-names  IDREFS  #IMPLIED
        >
```

Figure 4-9: Part of the BNDL grammar in the document type definition (dtd) format. The complete grammar is given in appendix A.

The drive element has precondition elements (mostly one that defines the state that the drive is motivating to satisfy). The *attncodelet* is an element to define an attention codelet, which may be explicitly programmed to watch a drive. Alternatively, an attention codelet can be specified by name in the attribute "codelet-names." The alias element allows a developer to give alias descriptions to elements; this enhances the explanatory knowledge about each element. The drive element also has attributes to define *importance,* and *intensity.* The intensity or urgency of a drive may change at run time, and this attribute is used just to initialize it. Also, the satisfaction attribute is used to specify the mode of satisfaction of the drive: once, continuous, or periodic. This attribute determines how the drive spreads motivational activation after being satisfied.

The BNDL can be considered as ontology for behavior networks. That is, it provides a common vocabulary with which to represent behavior network knowledge of a given domain. Sharing common understanding of the structure of information among people or software agents is one of the common goals in developing ontologies (Musen, 1992; Gruber, 1993). Particularly the BNDL is ontology for behavior networks in the sense of: (a) enabling reuse of domain knowledge, (b) separating domain knowledge from the operational/processing knowledge, (c) making domain assumptions or defaults explicit - such is avoiding of hard coding global parameters, and (d) possibly, analyzing or debugging domain knowledge.

4.6.2.2 BNDL Parser

The BNDL is used by an agent developer to specify the behavior network knowledge of a domain. The BNDL parser (BNDLParser.class) is the domain independent program in the behavior network module that compiles the BNDL specification (XML format) and produces instantiable behavior network components. The BNDL parser can decompile - it can take instantiable behavior network components and output the corresponding behavior network specification (XML format). The later capability is useful if an agent system is adaptive and makes changes in behavior network system. For instance, an agent with non-routine problem solving capacity could create new behaviors or new behavior streams; such creation of new components changes the behavior network system. The decompiling capability can be used to transfer the experience or knowledge specific to the behavior network system from one instance of agent to another agent that operates in the same or similar domain. Another use of decompiling capability is for debugging purposes in a development and deployment phases of an agent. An agent developer can imagine different ways of using this capability.

4.6.3 BNManager

BNManager (BehaviorSelector.class) implements the function of the skybox metaphor. This is where the activation/inhibition dynamics of instantiated streams takes place. For the most part, the behavior network computation happens as a result of local decisions of competence modules. But the algorithm has the following functions with global access of the instantiated streams: (a) normalization of activation levels so that the average activation level of nodes is bound by the value of the global parameter π, and (b) selection of the active behavior. These global computations happen in the *BNManager*. It is a thread that performs the following loop.

1. Sleep for a short length of time.

2. If no stream is instantiated, go to step 1.

3. Compute the total activation energy and the total number of nodes in the instantiated streams and make the results available for streams so that they can normalize the activation level of nodes in them.

4. Wait for stream level winners. If none found, go to step 6.

5. Select a stream level winner with the highest activation level and declares it a global winner to streams.

6. Synchronize with the instantiated streams for a new cycle.

4.6.4 Behavior Streams

Behavior streams (BNBaseStream.class) are partial order action plans that can handle a given problem or sub-problem. They can be defined with any number of behaviors and with or without any number of explicit goals. This is also implemented as a thread. It initializes itself by starting its goals and behaviors, and by registering itself to the list of instantiated streams. Then it performs the following loop.

1. Sleep for a short length of time.

2. Wait for the completion of the spreading of activation/inhibition in the stream.

3. Set normalized activation levels for its goals and behaviors.

4. Select stream level winner. If there are two or more executable behaviors with an activation level over the threshold, they will compete here. If there is only one, it is selected as a stream level winner. If there is no stream level winner, then go to step 9.

5. Make the stream level winner compete globally.

6. If the stream level winner is not the global winner, go to step 9.

7. Send the action event to its global winner behavior to perform the agent's action.

8. Wait for behavior action to complete.

9. Synchronize for new cycle with its behaviors and goals, as well as the other instantiated streams.

10. Check stream decay. If decayed enough, uninstantiate stream.

4.6.5 Behaviors

Behaviors (Behavior.class) are the fundamental object in the behavior network. They are also implemented as a thread. We have already discussed that behaviors are underlay by behavior codelets for their action and by expectation codelets for action feedback. They are also computationally defined in two different levels of abstraction. In one level, as part of the behavior network and, when instantiated, they take part in the network dynamics (skybox). And, in the second level, as a goal context – a coalition of processors in the form of behavior codelets and expectation codelets, take part in the dynamics of codelets – in the sense of pandemonium theory (Jackson, 1987). In implementation, the two abstract levels are separated. While the Behavior.class implements the role in the skybox, RepresentativeCodelet.class implements its codelet level functions, which will discuss it in section 4.6.4.

1. The behavior object has the following initialization steps.

2. Get relevant proposition objects in the state list manager.

3. Register itself to each of the proposition objects with the defined type of relationship (precondition, add and delete).

4. Build the different links that point to other behaviors and goals. The link type is defined by the relationship the source and destination objects have to the proposition associated with the link.

5. Start its codelet-level representative (RepresentativeCodelet.class), which, in turn, instantiates behavior and expectation codelets, if they are not already instantiated.

The behavior object performs the following loop.

1. Sleep for a short length of time.

2. If new links should be created or unused links should be destroyed, do so.

3. If signaled to die by decayed or satisfied streams, uninstantiate self.

4. Wait for completion of spreading of external activation from goals and behavior codelets with priming mode.
5. Spread activation/inhibition to linked behaviors and goals.
6. Wait for normalization of its activation level by the stream it is in.
7. Check its executability. If executable, register self for competition through its stream.
8. If lost competition, go to step 1.
9. Signal its behaviors codelets to start action via the representative codelet.
10. When behavior codelets' action is completed, report this fact to its stream.
11. Set its activation level to zero.

4.6.6 Representative Codelet

We have already mentioned that the representative codelet (RepresentativeCodelet.class) provides the behavior information and the role at the codelet level. After a stream and behaviors get uninstantiated, the associated codelets stay around until they decay. The representative codelet, representing its behavior, remains with the other codelets. If the behavior is reinstantiated before the representative and its other codelets decay, the behavior will be reconnected to the same codelets. In the initialization step (a) if instantiated instances are not found, it gets instances of behavior codelets from the corresponding generator behavior codelets, (b) it registers itself with the side line manager, and (c) it instantiates expectation codelets. Then it performs the following loop.

1. Sleep for a short length of time.
2. If there is no action event from behavior, go to step 1.
3. Signal expectation codelets for a new action cycle so that expectation codelets can start monitoring action by behavior codelets.
4. Signal behavior codelets to start action.
5. When the action is complete, report the fact to its behavior.
6. Check if it has decayed enough. If so terminate self.

4.6.7 State Items

State Items (State.class) are objects that represent the propositions defined in the precondition, add, and delete lists of behaviors and goals, and in the precondition lists of drives. All state items are stored in the state list manager. Each state item runs as a thread and starts with the initialization steps of (a) initializing lists to store goals and behavior objects that add and delete its proposition, (b) initializing lists to store drives, goals, and behaviors that are preconditioned by its proposition, and (c) adding itself to the *state list manager*. Then each state item thread performs the following loop.

1. Sleep for a short length of time.

2. If the lists that store the adding, deleting or preconditioned modules have changed, it signals relevant drives, goals and behaviors about the change. On receiving the signal, the drives, goals and behaviors check if they need to create new links and remove unused links.

4.6.8 Links

Link objects (Link.class) are also threads and implement all the three types of links. The link object knows the source and destination objects, the link type and typically the proposition that the link is associated with. The link object can gather the necessary information from the source and destination objects and computes the amount of activation/inhibition that needs to pass. Every time an activation event is received, the link object computes the activation/inhibition amount and sends it to the destination object.

4.6.9 Behavior Codelets

As discussed in section 4.3.2, a behavior codelet is the lowest construct to underlie a behavior. Its base class (BNBaseBehaviorCodelet.class) is a super class to all behavior codelet objects, which need to set initialization parameters and implement/override the actualExecution() abstract method. In its priming mode, a behavior codelet realizes IDA's cognitive cycle step 6 (CC-6) and as such it sense the relevance of "conscious" contents (situations or intentions) to its behavior; then its clone copy primes, instantiate and binds variables of relevant behavior streams (CC-7). The behavior codelet in priming mode does execute the following functions in a loop.

1. Sleep for a short while.

2. Sense relevant information from received broadcasts. If none found, go to step 1.

3. Process priming function: Priming could be forward processing when a behavior codelet is set for situation relevance. The priming could also be backward processing if a behavior codelet is set for intentional relevance (need-to-satisfy-proposition or is a goal image that need to be satisfied); this is a back-chaining of action planning and can be used in deliberative processing like Non-Routine Problem Solving (see chapter 6).

4. Generate instance of its clone with variable binding information as an independent thread; if there is one already, send the clone rebinding information. Here, the clone rebind primed behavior streams and then spread environmental activation to primed behavior streams.

A behavior codelet in action mode has the function of executing a behavioral action (CC-9) when the behavior it underlies is selected for action (CC-8). A behavior codelet in action mode is always a cloned instance and performs the following main steps in a loop.

1. Sleep for a short while.

2. If its behavior is active (signaled by representative codelet), join the playing filed and execute. Otherwise go to step 4.

3. Send completion of action to representative codelet.

4. After a fixed time window, leave playing field.

5. If its associations with all other codelets are decayed away (forgetting mechanism), uninstantiate/terminate self. Otherwise go to step 1.

4.6.10 Expectation Codelets

As already pointed out in section 4.3.4, an expectation codelets is a special attention codelet that realizes a type of anticipatory control to a behavior it underlies. Upon the selection of its behavior (CC-8), and along with its sibling behavior codelets, an expectation codelet acts to watch the effects of the behavioral action and compile feedback on the outcome, which may take multiple cognitive cycles. The feedback, if expectation codelet wins a competition for access to "consciousness," becomes a global broadcast content. A generic expectation codelet (BNBaseExpecattionCodelet.class) is implemented and its instance compiles relevant information from the knowledge built into its sibling behavior codelets. It performs the following loop as an independent thread.

1. Sleep for a short while.

2. If its behavior is active (signaled by representative codelet), continue; otherwise go to step 1.

3. Setup monitoring information – have knowledge of what to watch in the work space.

4. Join the playing filed and watch for relevant events in workspace until completion of execution of sibling behavior codelets is signaled (by representative codelet).

5. Compile action outcome - compare the expected result with the real outcome as observed in the workspace.

6. Compete for access to "consciousness" with the action outcome as content. The codelets' activation level, which depends on the automatization and deautomatization mechanisms (see chapter 5), determines its relative competitiveness.

7. After fixed time window, leave playing field.

8. If its associations with all other codelets are decayed away (forgetting mechanism), uninstantiate/terminate self. Otherwise go to step 1.

4.6.11 Implementation Architecture

The implementation architecture of the behavior network module is shown in Figure 4-8. The *domain independent* architecture provides a basic mechanism for the behavior network system. It also provides a frame work (a) to design and implement a domain specific action selection mechanism, (b) to interact with other domain independent cognitive modules, including "consciousness," different memories, and perception, (c) to provide an action selection base system to add new domain independent mechanisms such as problem solving and deliberation.

4.6.11.1 Domain Independent Components

The central part of the domain independent section of the architecture is the *behavior network starter,* which begins the action selection system. Its *initializer* component has multiple functions. First, it initiates the compilation of the XML program that defines the domain knowledge using the *BNDLParser* that uses Behavior Network Description Language (BNDL) grammar. The XML program contains the behavior network specification and it should satisfy the syntax of BNDL grammar. The compilation output defines the *behavior network template*. Second, the *initializer* starts the *side line manager*, the *codelet list manager*, the *state list manager*, the *BN workspace* module (which is part of the long-term working memory system), *BNManager*, *drives*, and others. At the end, the *initializer* populates the *state list manager* with state item objects and the *codelet list manager* with generator behavior codelets. The *initializer* uses the codelet loader program to create an instance of codelet objects out of the corresponding codelet class files. The codelet classes are coded by the application developer and must be available in a directory accessible to the virtual java machine. At this point, the behavior network architecture has been loaded with the necessary domain knowledge content and is ready to receive stream priming situations and start the action selection dynamics in the *BNManager*.

4.6.11.2 Domain Development

The domain independent behavior network architecture defines the mechanism for action selection. The personality, or characteristics of the action selection system, is explicitly defined when the domain specific content is bound to the architecture. Figure 4-10 shows how an agent developer uses the behavior network module. The tasks of the agent developer are: (a) to design behavior streams and describe it in an XML file, (b) to implement all the necessary codelets (behavior, expectation, attention, information), and (c) to implement an agent starting program that boots all the necessary agent modules in the right order. This task could be repeated in multiple development cycles.

Design Behavior Streams

First, the agent's domain should be specified in terms of the behavior network system: drives and streams (goals and behaviors). The relative importance of drives is specified, and one or more behavior streams can be designed to satisfy each drive. Additional behavior streams can be designed to handle sub-problems of the agent. For instance, the behavior node specification will start with identification (name and possibly aliases), precondition list, add list, delete list, behavior codelets, expectation codelets, and variables.

Once the specification of the drives, goals, and behaviors are completed and the behavior and expectation codelets are specified or possibly coded in previous development cycles, the next step is to write the specification in XML text file using the syntax provided by the *BNDL grammar*. The *BNDL Parser* (BNDLParser.class) program can be directly used to check if the XML file compiles correctly.

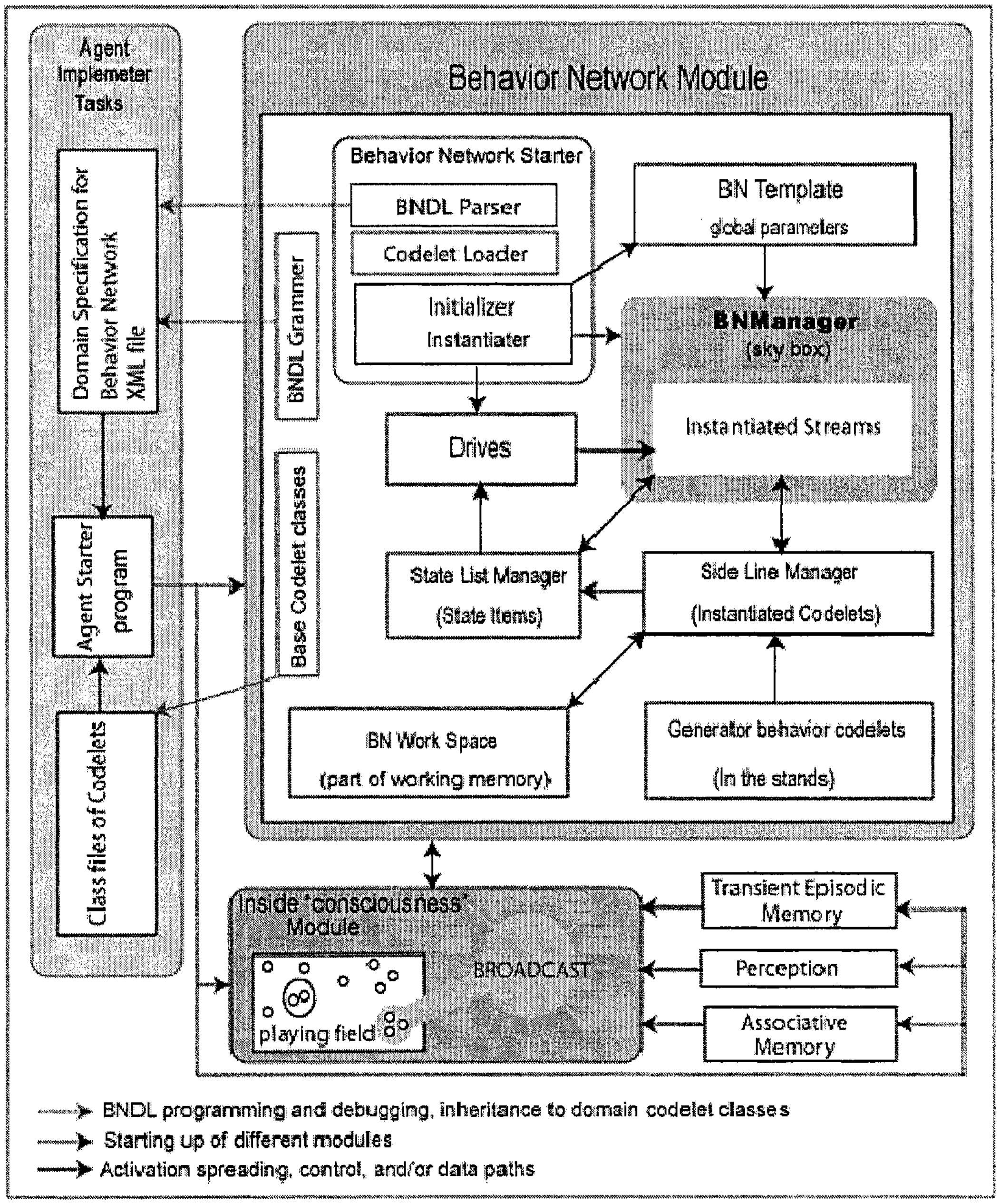

Figure 4-10: Interaction of the behavior network module with "consciousness" and some of IDA's other modules. The primary tasks of the domain implementer (in blue box) and their interaction with the different modules are also shown.

Coding Codelets

Once the behavior network is designed, then the next step is to implement its codelets. Its behavior codelets should implement the actual actions of the behavior. If a behavior has multiple behavior codelets, the action of each codelet should be mutually exclusive. (One does not depend on the other or they should not conflict in using the same resource). One or more expectation codelets implement the monitoring of actions of the sibling behavior codelets and reporting of the outcome.

In the behavior network framework, we provide base behavior and expectation codelet classes. The domain dependent behavior codelet classes and expectation codelet classes are sub-classes of the provided base behavior codelet class and base expectation codelet class respectively. To be more specific, a domain specific behavior codelet implementer should be aware of two things: (a) to state whether the behavior codelet is priming or non-priming in its constructor method and (b) to implement one abstract method of the base behavior codelet: *actualExecution()*. In this method, the binding values of variables associated with the precondition list are used as input information, and the effect should produce state change that can be used to change the binding values of variables associated with the add list.

Similarly, the domain specific expectation codelets inherit from the provided base expectation codelet class and implement a few abstract classes - particularly, methods to setup action monitoring conditions, to start action monitoring, to report successful outcomes, and to report error outcomes. A domain agent implementer may not need to code an expectation codelet all the time. There is a generic expectation codelet, which uses the structural knowledge of the behavior and activity pattern of sibling behaviors to perform the monitoring and reporting functions. The knowledge that the generic expectation codelets use includes the variables (their types and binding values) and the effect of the actions of the sibling behavior codelets in the long-term working memory. If explicit expectation codelets are not specified for a behavior in the XML file, then such generic expectation codelets will become active automatically during the instantiation of the behavior.

The domain application should also have attention codelets, which monitor the arrival of a specific situation. The specific situation may arise from anywhere in the agent system: in the behavior network module, in the perception module, problem solving module, or other modules. For the most part, internal and environmental situations are reflected in the long-term working memory, and that is where most of the monitoring by attention codelets takes place. In relation to the action selection system, two different base attention codelets are implemented. One is relevant to watching behavior and goal nodes in instantiated behavior streams. The second is relevant to watching interesting changes in the behavior network (BN) workspace. The BN workspace is logically part of the long-term working memory of the IDA model. Both types of base attention codelets are complex. The complexity arises mainly from the implementation requirement that the author set for generality. As with any of the other codelet types, domain specific attention codelets can be implemented by inheriting these base attention codelets.

Agent Booting Program

The booting of the agent system should start all the attention codelets along with the "consciousness", behavior network, perception and other modules. The XML file specifies the required domain knowledge to start the behavior network module. The agent booting programs should start and initialize the different agent modules in the right sequence. Assuming the perception and memory systems are implemented for a domain, Figure 4-10 shows the type of interactions a domain implementer will have with the behavior network module and other modules.

4.7 Testing and Results

Our action selection mechanism is based Maes' behavior network (1989). While we added a number of important capabilities on her mechanism, part of our mechanism fully inherits her mechanism and its properties. Particularly, our skybox component has the same activation/inhibition spreading dynamics and global control parameters as that in Maes' mechanism but with the addition of importance parameters for drives and discrimination-factor global parameter. The skybox is where the behavior network action selection dynamic happens and it contains all the instantiated behavior streams (their variables are bound, receive environmental activation from behavior codelets, and receive motivation from drives). The only major difference is on the extent of instantiation of component modules of the system. Maes' mechanism has all its competence modules instantiated all the time, although many of the competence modules may be irrelevant in a given situation. But our mechanism in the skybox has only instantiated behavior streams that are solutions to problems at hand. That is, irrelevant behavior streams are not instantiated. Also, the set of instantiated behavior streams vary from time to time. Maes (1989) tested her behavior network mechanism and reported the results. We do not intend to reproduce her tests or make similar tests. Our objective is to test how well the action selection and the "consciousness" modules work together in a realistic domain. Particularly we (i) present GOMS analysis of domain tasks, (ii) demonstrate how importance and discrimination-factor parameters enable the setting and maintenance of relative priority among competing goal context hierarchies, and (iii) test how well the action selection mechanism as a robot controller perform compared to a human operator in executing the same domain task.

Figure 4-10 shows high-level interaction between the behavior network module and "consciousness" system. As discussed in the cognitive cycle, the conscious broadcast recruits goal context hierarchies (behavior streams), and the behavior network system implements the cognitive cycle (CC) steps 6 to 9. The action taken at the step 9 of the cognitive cycle may influence what comes to "consciousness" in the future. This is the interleaving of conscious content and the unconscious goal context (and its action). This is depicted in Figure 4-7 as a C-U-C triad: *conscious experience* followed by an action of an *unconscious dominant goal context* (selected behavior) and followed by another *conscious experience*. Our test is to show (a) how the behavior network module and the "consciousness" module interact properly, (b) if the interaction exhibits the C-U-C triad, and (c) how multiple goal context hierarchies compete and interleave each other. The conscious experience could arise as a result of being aware of the effects of a performed action that may provide novel and specific information (used to bind variables of behaviors) for the actions to come.

4.7.1 Agent, Domain and Environment Setup

4.7.1.1 Domain Tasks

We will show IDA's action selection mechanism controlling a robot in a warehouse domain, where it performs different tasks in a given arrangement on the warehouse floor. The warehouse stores different items. The robot has the tasks of unloading items in unloading areas and shelving them in shelving areas. When orders to ship items are received, the robot un-shelves and put them in the designated shipping area. Each item type will have designated unloading and shelving areas. In the warehouse, an item should be shelved before it could be shipped; this could be considered a requirement for inventory and other processes that the robot does not perform. In this domain, the robot moves items around and maintains its energy level. It will have three main drives: (i) go to feeding area for recharging, (ii) unload items and put them in their designated shelving areas, and (iii) take items from shelving areas to their designated shipping areas. To satisfy each drive, the robot may have one or more behavior streams. Each behavior stream corresponds to a task or subtask that the robot should perform. The robot moves items in the warehouse using well-defined paths and can carry only one item at a time. The robot's energy level goes down with the number of steps it takes to move, and it needs to recharge before its energy level nears zero. Also the robot will be able to perceive its world completely at all times. This domain allows us to adjust the robot's world with the required level of complexity. The objective here is to demonstrate how the action selection module works with the "consciousness" module in controlling an agent with multiple drives and behavior streams that compete with each other and to show effects of some of the control parameters of the behavior net on the decision making dynamics.

4.7.1.2 General Environment

Robot

In this section we show how IDA's action selection mechanism controls an autonomous agent in the warehouse domain. Particularly, we use a Khepera robot in a simulation environment. The Khepera robot from K-Team (http://www.k-team.com/) is a small mobile robot that is used in many robotic control related researches. The Khepera robot has a number of free and commercial simulators and we use the freeware WSU Simulator from Wright State University (http://cs.wright.edu).

The base Khepera robot has two differential wheels and eight infrared sensors that are distributed around its perimeter (Figure 4-11). Each infrared sensor gives two types of data readings: object proximity (distance) and light intensity. The robot can have additional turrets such as gripper and video camera.

According to the developers of the simulator, the WSU Khepera robot simulator is designed to mimic a specific robotic hardware configuration and testing environment in their research lab. That is, the WSU Simulator system consists of a standard Khepera robot with its gripper-arm turret extension and a 4'X 4' enclosed arena that may contain walls, lights, caps and/or balls. Walls and lights are assumed to be static objects in that they cannot be affected in any way by the robot. Thus if the robot were to drive into one of these objects, it would stop and possibly become stuck. Caps and balls are considered dynamic objects; they will move if the robot makes unintentional contact with them, and the robot's gripper can directly manipulate them. Besides the infrared sensors, wheel encoders determine robot position and direction, and the gripper object sensor indicates if there is an object between the grippers. The effectors are wheel motor speed, arm position (up or down), and gripper state (open or close). The robot can have only one item between its grippers.

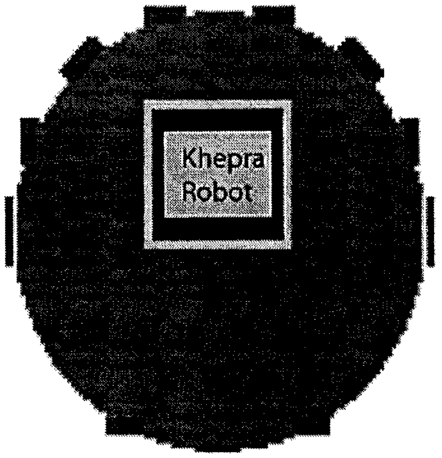

Figure 4-11: Khepera robot with the position of its 2 differential wheels and 8 infra-red proximity and ambient light sensors.

The WSU Khepera robot simulator system is modified with additional coding to add capabilities suitable for our research needs. The first addition is a supervisor functionality that gives the simulator system the ability to sense the real-time position (GPS information) of objects that are present in the simulator environment and to send the information to the robot controllers. In the real Khepera robot, a supervisor can have remote communication via a radio turret. Using the GPS information, the controller has complete perception on the position of objects. The second addition is to have a robot charging/recharging capability in the simulator. In relation to charging, the robot can sense the available energy level and as well as the total energy it has used since it has been in use.

Base Environment

The robot operates in a world defined in the simulator's enclosed arena and its controlling behavior depends on the dynamic situation – the internal state of the robot plus the positions of the robot and all other objects in the arena. The warehouse floor plan is shown in figure 4-12. Different areas on the floor have different utilities: unloading, shelving, shipping, charging and road areas. The road area itself is conceptually divided into blocks: unloading, shelving, and shipping. Item-1 types (balls) and item-2 types (caps) have designated unloading and shelving areas. An order can have items of both types and could be shipped at either of the shipping stations. There are two charging stations and they may or may not have a charger. A light object is designated to be a charger. Static information such as the floor plan and designated processing points such as shelving and shipping points are stored as world knowledge, which is available to the controller.

The robot controller has been provided with built-in high-level percepts that combine multiple primitive sensory inputs. There are also built-in percepts that input facts in the world such as the position of different objects in the different processing areas of the warehouse floor. For instance, percepts could assert facts such as "two of item-1 types in the shelving area" or "charger exits in charging station at north."

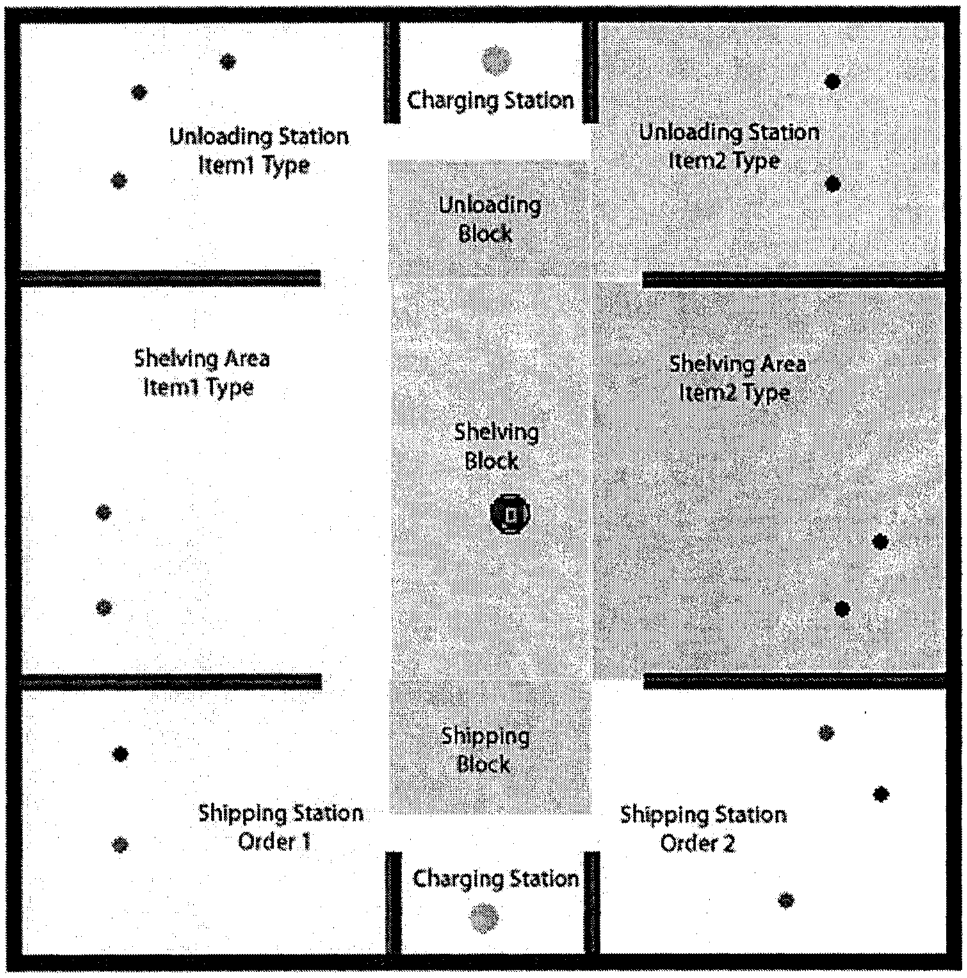

Figure 4-12: Warehouse floor plan and designated areas of activities.

Due to heavy computational needs, the robot simulation system and our robot controller could not run on the same personal computer. So we needed to setup a remote controlling computational environment in which each program module ran on different computers. To realize the remote control, a client/server based custom network communication module was implemented. This module allows the controller system to pass different control commands to the robot simulation system and the robot simulation system to pass environmental information to the controller system.

The diagram in figure 4-13 depicts the multi-level computational structure of the robotic agent. The high-level control is implemented as the "consciously" mediated behavior network in the IDA framework. Since our research objective addresses IDA's decision making mechanism, the controller does not encompass all possibly relevant IDA modules such as the associative memories and the perception mechanism in its sophisticated form. The "perception" in this controller's sense is a simple passing of environmental information and that is without any cognitive filtering. Percepts are directly copied to the long-term working memory and from there onwards the standard cognitive steps of IDA proceed as we will see below.

A selected behavior controls the robot by the actions of its behavior codelets, which invoke actions as high-level commands such as for control of motions and manipulation of arm and grippers. To process high level commands, a device driver type of program is added to the robot simulator system; the program interfaces the controller with the base robot and its environment. It could also be considered as a robot controlling macro program. In top-down direction, it translates the high-level commands to organized sequence of relevant robot actuator values and the opposite direction it provides command feedback and derived sensory information as percepts to the controlling agent. The driver/macro program sets sequence of actuator values based on what is sensed from the environment (external sensory input) and based on the state of the robot (internal sensory input). The robot effects its environment as per the set actuator values and towards satisfying the current high-level command. At the end of a sequence of activations of actuators, derived sensory information and a feedback for the high-level command is made available for the robot controller.

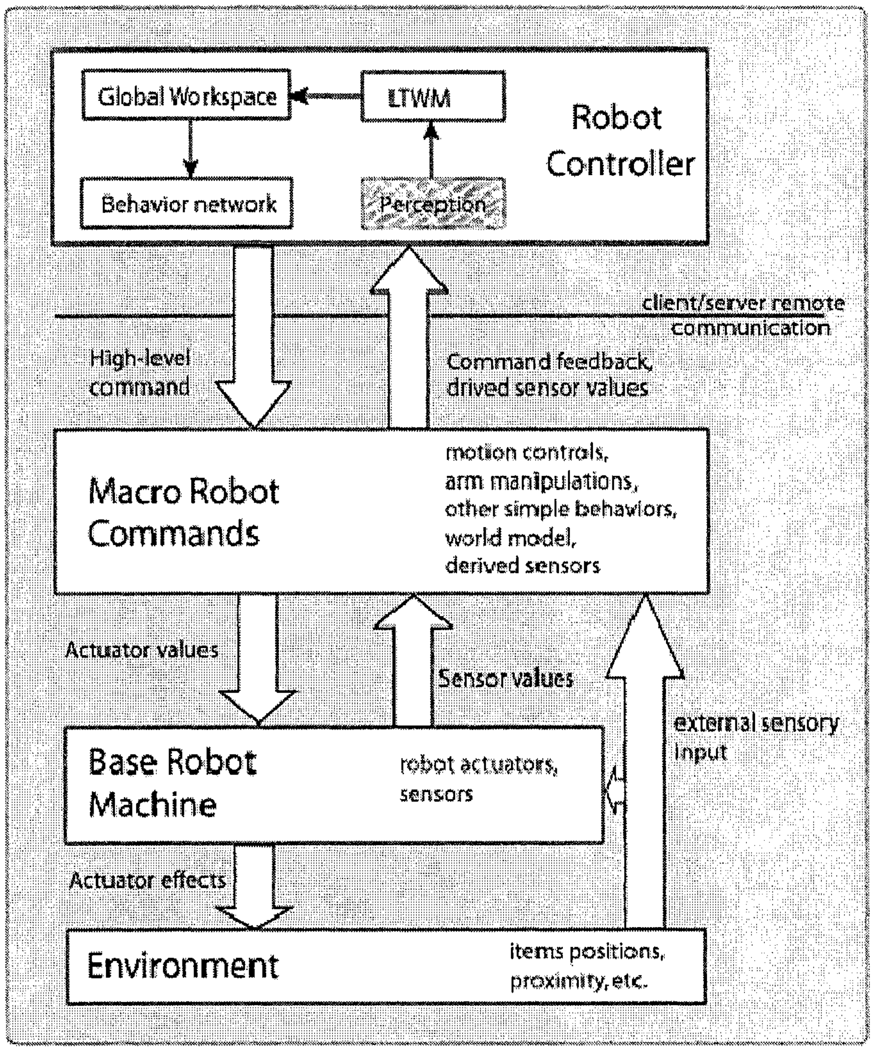

Figure 4-13: Multi-level structure of the robot control.

The computational architecture (figure 4-13) shows how high-level controller communicates to the robot system over a remote communication medium. In general, a remote communication could be set to happen at any one of the levels in the structure. But, our approach keeps the high-level macro command programs at the robot side (rather than at the behavior codelet side) and this has advantages particularly when the control needs to be remote, high-level, and with relatively high communication overhead compared to computational overhead.

4.7.1.3 Behavior Streams for the Domain Tasks

As discussed in section 4.6.7, the implementation of the IDA's action selection system starts with the design of behavior streams that realize the tasks that need to be performed in a given domain. Depending on the level of abstraction for behaviors, different behavior streams can be designed to do the same task. Our objective in the experiments is (i) to demonstrate action selection or decision making as a plausible cognitive process in executing tasks and (ii) to show the effects of some of the external control parameters of the behavior net towards tuning the dynamics for a required decision making property for the given domain tasks. Both could be shown at any abstract level of decision making. For the warehouse domain, we designed the behavior streams so as to keep them at a higher level of abstraction. If we take the tasks of unloading and shelving type-1 items and suppose that the robot is in the middle of the floor (at its resting point), then we can design a behavior stream for the task of unloading and another behavior stream for the task of shelving. The execution of these behavior streams in the corresponding order will move the item from the unloading area to its shelving area. Table 4-2 shows the behavior stream that unloads the type-1 items. For this domain, we specify behavior streams to unload, to shelve, to unshelve, and to ship the type-1 and type-2 items. We also specify a behavior stream to charge the robot at the nearest charging point when its energy level is below a given threshold. Then the specification is transformed into behavior net definition language and written as an XML file. Part of the XML shown in Appendix B specifies the behavior stream discussed in table 4-2.

Table 4-2: Role of each behavior in the "unload item" behavior stream.

Behavior	Role
Get Unloading Area	Checks if this task should be started: (i) if item-1 type is in the unloading area of item-1 type, (ii) if there is empty shelving space available for item-1 type, (iii) if gripper is empty so that it could be used to pickup, and (iv) if this task has not been already initialized. It has the effect of asserting (to predecessor behaviors) readiness to unload in the unloading area of item-1 type and the initialization to unload item-1 type. This behavior does not control the effectors of the robot.
Move to Unloading Block	Executes to move the robot from its current road position to the unloading block. That is, if the robot is not already there, gripper is empty, and unloading task for item-1 type is started. After successful execution of its behavior codelets, the effect of this behavior is to assert that it is in the unloading block and not in the other blocks.
Move to Unloading Entry	Executes if the robot is in the unloading block, gripper is empty and unloading task for item-1 type is started. Its execution moves the robot to the entry point of the unloading area for item-1 type. Its effect asserts that the robot is at the entry point and the robot is no longer in the unloading block but in the unloading function area.
Locate Item to Unload	Executes if the robot is at the entry point, unloading for item-1 type is started, and gripper is empty. Its execution selects one of the item-1 types that need to be unloaded and the effect is to assert that a specific item-1 type is selected for unloading. This behavior does not control the effectors of the robot.
Move to Item to Unload	Executes if item-1 type instance is selected for unloading, gripper is empty and unloading for item-1 type has started. Its execution moves the robot near to the selected item and asserts that it is near to the item and it is no longer at the entry point.
Pickup Item to Unload	Executes if robot is near the item to be unloaded and its gripper is empty. Its execution is to approach the item with its arm towards the item, and to lower its arm, to open its grippers and move until the item is between the grippers, and to pick up the item by closing the grippers and raising its arm. Its effect asserts that item unloading is done and gripper is in use.
Exit Unloading Area	Executes if item unloading is done. Its execution moves the robot from the item-1 type unloading area to the unloading block area. Its effect asserts that it is at the unloading block.

4.7.1.4 Implemented Codelets

Attention Codelets

The warehouse robotic agent has a complete perception of its environment all the time. The robot does not use IDA's perceptual and memory (associative and episodic) modules. Every perceptual input of the robot goes directly to the long-term working memory. Some percepts are direct sensor information like "gripper empty" status, and others are derived from multiple sensory and world knowledge pieces like "number of available shelving points in an area." As we discussed before, the content of working memory changes as a result of new perception or direct behavioral actions. Attention codelets become active when they find relevant information in the working memory segment they are watching. The pieces of such information include "low battery energy," "gripper empty," and "item needed for shipping." The warehouse domain needs about two dozen attention codelets, each monitoring for specific content in the long-term working memory. Active coalitions of attention codelets compete for access to "consciousness" (CC-4). When and if a coalition wins the competition, its content is broadcast (CC-5).

Behavior Codelets with Priming Mode

Behavior codelets (with priming mode), realize IDA's cognitive step six (CC-6). Each responds to relevant information in broadcasts, and an instance binds its variables with the received piece of information from the broadcast.

A behavior stream is primed, instantiated and its behavior bound by one or more of its underlying codelets (CC-7). Referring to the item unloading behavior stream (table 4-2), the stream is underlain by three different instances of behavior codelets with priming mode: (i) a behavior codelet that is relevant to the availability of item type in the unloading area, (ii) a behavior codelet that is relevant to the availability of shelving space for the same item type, and (iii) a behavior codelet that is relevant to the emptiness of the robot grippers. Instances of the first two behavior codelets underlie the "Get Unloading Area" behavior, and the instances of the third behavior codelet underlie all the behaviors that need "empty gripper" status for their executability. This behavior will not start to execute until all the three instance of these codelets prime and bind their underlying behaviors.

Behavior Codelets with Action Mode

Such behaviors encode the actual action of behaviors they underlie. After a behavior is selected to control the robot (CC-8), its underlying behavior codelets perform the actual behavioral action (CC-9), which changes the internal state of and/or the external state of the robot. Referring to the "unload item" behavior stream (Table 4-2), the execution of the behaviors "Get Unloading Area" and "Locate Item to Unload" make changes only to internal state, and the rest make changes to the external state (use some effectors of the robot).

Below (table 4-3), we discuss the high-level robot control commands that are available for the different behavior codelets that control motor actions - robot's movements and its many manipulations in the environment. Behavior codelets are relatively high-level constructs. That is, at each execution, a behavior codelet issued commands to control the robot for a relatively long time and possibly to activate multiple effectors. For instance, a behavior codelet that has a goal "pickup item" does all the following: forward movement to get close to the item that needs to be picked up, open grippers, down the arm, move forward until item is between grippers, close grippers, raise arm, and move backward and away from the pickup point. One can change the granularity of behaviors and behavior codelets, but the granularity level of behaviors and behavior codelets will not affect the decision making process, which is our research objective.

Table 4-3: High-level robot commands that can be invoked by behavior codelets.

Behavior Codelet's task	Parameters	Description
MOVE TO A POINT	Destination point	Moves the robot in the right direction until it reaches the destination point.
MOVE TO A BLOCK	Block name	Moves the robot towards a point in the named block of the floor until it reaches the point.
MOVE TO ENTRY	Processing area name	Turns the robot in the right direction and moves to the entry point of the given processing area.
MOVE TO ITEM	Item ID	Moves towards the item of a given ID and stops near to the item.
PICKUP ITEM	Item ID	Controls the robot so that it approaches and picks up the near by item with a given ID. It first approaches the item slowly until the robot front distance sensors show closeness. Then it opens its grippers, lowers its arm and gets closer until the item is in its open grippers. Then closing its grippers, the robot raises its arms, and moves a little in the backward direction away from the pickup point.
DROP ITEM	Drop point	Move the robot in the front to the dropping point. Then it lowers the arm, open the grippers, move backward a little, and raise the arm.
APPROACH CHARGER	Charging station area	Moves the robot towards the charger, which is represented by a light source object placed in the charging area. The movement is guided by the direction of strongest light intensity.
CHARGE		Moves the robot closer to the charger and charges its battery. The progress of the charging is indicated by a wiggling movement of the robot. When fully charged, the robot moves a little away from the charger.

Behavior codelets perform their given tasks using the values of their bound variables as the values of the task parameters. Variables bound by the time preconditions are satisfied, after which execution is possible.

4.7.2 GOMS Model Analysis for Warehouse Robot Tasks

The IDA's action selection mechanism (IDA-ASM), as the robot controller in the warehouse domain, performs primitive (codelet level) actions to execute its domain tasks. The primitive action could be internal (cognitive) or external (observable behavior that activates the effectors of the robot). We have already discussed the role of the action selection module in IDA's cognitive cycle. As a robot controller application in a warehouse domain, we can also predict and evaluate the sequence of actions and execution time in performing the given domain tasks using one of the GOMS task analysis techniques (Kieras, 1996, 2003).

GOMS is an acronym for Goals, Operators, Methods, and Selection rules. John and Kieras (1996a) define the GOMS concept as being useful to analyze the knowledge of how to do a task in terms of goals, operators, methods, and selection rules. GOMS model is a description of procedural knowledge that a cognitive agent must have in order to perform tasks on a machine or system. *Goals* are what a cognitive agent has to accomplish. *Operators* are actions performed to satisfy the goals. Operators are primitive actions, which could be perceptual, cognitive, motor, or composites of these. Operators, when executed, can change internal states in the cognitive agent and/or the external state in the environment. The execution times of operators must be predetermined and could be approximated by a constant (like average), by a probability distribution, or by a function of some parameter. Execution time and other operator parameters are assumed to be independent of the context under which the operator is used. *Methods* are sequences of operators that are performed to accomplish goals. Methods can call subgoals, which need to be accomplished by other methods. So methods have hierarchical structures. Methods describe tasks that need to be performed and the description content depends on the possible operators. In a given domain context, there may be more that one method to accomplish a goal and the *selection rule* chooses the appropriate method for the context. GOMS modeling starts after task analysis, which is a task decomposition process using techniques such as Hierarchical Task Analysis – HTA (Annet & Duncan, 1967). The hierarchical decomposition is employed until primitive operators are reached or when the decomposition reaches a chosen granularity. Operators are at the lowest level of detail that can not be broken down further. In the task analysis, the goals, agent's state and working context must be specified. Then the GOMS analysis provides the details of how the agent does the tasks. As Kiares (1996) puts it, GOMS gives the analysis of procedural knowledge that describes the answer for "how to do it" questions. GOMS analysis gives an accurate prediction on the assumption that tasks are performed by procedures represented as routine cognitive skills.

The GOMS family consists of four variants (the Keystroke-Level Model (KLM), the original or CMN-GOMS formulation, NGOMSL, and CPM-GOMS). The details are presented by John and Kieras (1996a, 1996b). Among all, the NGOMSL is the most structured well specified technique. Our GOMS analysis will use the NGOMSL version and we will discuss it in more detail below.

4.7.2.1 NGOMSL

NGOMSL is an acronym for **N**atural **GOMS L**anguage, which is a structured, program form natural language used to represent the GOMS model or the procedure that must be performed (Kieras, 1997). The NGOMSL model can predict operator sequence, execution time, and time required to learn methods. The NGOMSL analysis is preceded by task analysis. Then, with a natural flow from the task analysis, the model performs a top-down, breadth-first expansion of the top-level goals into methods until all low-level methods contains only primitive operators.

The strength of the NGOMSL model is its embracing of a cognitive architecture called "Cognitive Complexity Theory" (CCT) (Butler et al., 1989). CCT is a serial-stage cognitive architecture in which a working memory event triggers a production rule that applies at a fixed rate. The execution of these rules produces internal (change of contents in working memory) or external (execution of external primitive operation like keystroke) effects. NGOMSL associates GOMS concept with CCT format by representing GOMS methods with sets of production rules in a prescribed format. Each production rule is a step in a procedure represented by a method. NGOMSL explicitly represents the procedure with hierarchical and sequential methods. Perceptual and motor activities are represented by external primitive operators. Internal operators represent cognitive steps of the CCT architectural mechanism, such as adding/removing information in working memory, asserting goals to be accomplished or asserting status of goal accomplishment.

Although our objective is to predict the execution time of the domain tasks, CCT provides good predictions of both execution time and procedure learning time (Bovair et al., 1990) and so does GOMSL. As Kieras (1997) points out, the time to execute a method depends on the time needed to execute the operators and on the number of cognitive steps or production rules involved. Each asserted statement in NGOMSL format corresponds to a production rule that is executed. The execution time can only be estimated for specific task instances in which one can determine the number and sequence of steps and operators. Execution time of a task instance is the sum of the time taken by all production rules, time taken by all external primitive operators, time taken by all custom operators, and time taken for waiting system responses. Each production rule takes 0.1 seconds; each primitive external operator takes a predetermined time; each mental operator takes an estimated time of 1.2 seconds or a predetermined time set by the analyst. The predicted time required to learn how to perform an instance task (or pure learning time) is the sum of the *pure method learning time* and the Long-term Memory *(LTM) item learning time*. Pure Method learning time is linearly proportional to the number of production rules (NGOMSL statements) in the method that had to be learned (Bovair, et al. 1990). LTM item learning time is the time required to memorize items that will be retrieved from LTM during method execution.

GOMS analyses could be done manually, but we use a GOMS model simulation tool called "quick and dirty" GOMS or QGOMS, which allows rapid model construction and analysis (Beard et al., 1996).

4.7.2.2 NGOMSL Analysis of a Domain Task

GOMS modeling starts with a task analysis that itself is based on two issues: (i) the description of the world, which we have already explained, the environment and how the controller accesses it; and (ii) an account of how the task is performed in the described world. The task analysis (task decomposition) is already addressed by the specification of the behavior streams (action plans for the different tasks). As the case of GOMS concept, behavior stream specification could be considered to be a type of hierarchical task analysis. If we make the components of the behavior net and the GOMS concept correspond with each other, goal states can map to goals, streams can map to structure of high-level methods, behavior nodes can map to low-level methods, and behavior codelets can map to operators. Goal nodes are the same as behavior nodes without underlying behavior codelets and they map to methods with no primitive operators. GOMS concept explicitly specifies selection rules if there are multiple methods to achieve the same goal/subgoal. In the behavior net, selection rule is not explicit; there may be multiple behaviors to satisfy the same goal/subgoal. If there are such behaviors, a corresponding selection rule can be generated in the GOMS formulation. If there are two behaviors to satisfy a proposition, then the GOMS analysis should have a selection rule to choose between the two behaviors in satisfying the proposition as a goal. So behavior stream specification gives us the required task analysis to our NGOMSL modeling.

Table 4-4 shows the manual NGOMSL analysis for the "unload item" task we discussed in table 4-2 above. As we can see, the goal and the behaviors in the behavior stream are shown as NGOMSL statements. Under each method of a behavior, there are multiple primitive operators. A controlling behavior activates its underlying behavior and expectation codelets. At pre-execution of a behavior codelet, a number of mental operations take place (like triggering/selection of a behavior and checking of executability) and at post-execution of a behavior codelet, more mental operations take place (like providing feedback on the outcome of the action). So, under each method of a behavior, we have: (i) pre-action mental primitive operators, (ii) the actual motor or mental primitive operator that perform the task of the underlying behavior codelets, and (iii) post-action mental primitive operators. The primitive operators under a behavior method execute sequentially in a given order.

The time required to perform mental operations including perception varies and depends on the cognitive processes involved in the operation. A good overall estimate for the duration of mental operators is 1.2 seconds (Oslon & Oslon, 1990). As Kieras (1997) pointed out, each NGOMSL statement incurs in 0.1 seconds execution time, and whatever associated operator times involved are additional. In NGOMSL analysis, by default the execution time of each built-in primitive mental operator is bundled into the execution time of the NGOMSL statement. The 0.1 seconds execution time is used in the assumption that each NGOMSL statement is actually realized as a single production rule. The exact timing of mental operators is not critical in our analysis since our focus is mainly to explain and account for the plausible cognitive process involved in the warehouse tasks. So we will use the 0.1 seconds duration for all primitive mental operators. Looking at the methods analyses in table 4-4, the "Get unloading area for item" and "Select item to unload" methods have only primitive mental operators and each will have duration of 0.1 seconds. Each in the rest of the methods has primitive operators that interact with the robot, namely, issuing a command to the robot and waiting for feedback on the issued command from the robot. Issuing a command is considered a mental operator, and it also has an estimated duration of 0.1 seconds. The waiting time for feedback varies based on the command passed to the robot. As we have discussed earlier, all primitive operators should be fully characterized before analysis starts. As such, the execution time for waiting primitive operators should be predetermined. This was done using data from experiments performed by a human operator controlling the robot in the same simulation system.

Referring to table 4-4, the "Get unloading area for item" behavioral method has five primitive mental operators. The "Think of the item-type for the task" operator helps to access an "item-type," which is a parameter to the "unload item" task. The operators "Determine unloading area for item-type" and "Retain unloading area for item-type" are the main behavioral actions to decide the unloading area for the item-type and to store the area information in working memory for later use during the execution of the "unloading task." The operator "Verify goal satisfaction status" includes the cognitive process associated with the monitoring and action feedback reporting functions of expectation codelets; this operator may include perception of the internal state and the environment. The last operator "Return with goal accomplishment" is recommended by NGOMSL modeling (its strong program form) to have one such last operator under every method to assert returning of the goal accomplishment state.

Table 4-4: A manual NGOMSL analysis for "unload item" task.

NGOMSL methods for *Unload Item* task	Execs	Ext. Op. time
Method for goal: Get unloading area for item	1	0.1
Step 1. Think of the item-type for the task	1	0.1
Step 2.DETERMINE unloading area for item-type	1	0.1
Step 3.RETAIN unloading area for item-type & VERIFY goal satisfaction status	2	0.2
Step 4.RETURN with goal accomplishment	1	0.1
Method for goal: Move to block of unloading area for item	1	0.1
Step 1.RECALL unloading area for item	1	0.1
Step 2.DETERMINE the unloading-block area for item	1	0.1
Step 3.ISSUE robot command to move to unloading-block area	1	0.1
Step 4.WAIT for response of robot	1	104.8
Step 5.VERIFY robot is at the unloading-block area (goal status)	1	0.1
Step 6.RETURN with goal accomplishment	1	0.1
Method for goal: Move to entry of unloading-area for item	1	0.1
Step 1.VERIFY robot is at unloading block	1	0.1
Step 2. RECALL unloading area for item	1	0.1
Step 3.DETERMINE entry point to unloading area	1	0.1
Step 4.ISSUE robot command to move to the entry point	1	0.1
Step 5.WAIT for response of robot	1	61.2
Step 6.VERIFY robot is at entry point & RETAIN entry point to unloading area	2	0.2
Step 7.RETURN with goal accomplishment	1	0.1
Method for goal: SELECT item to unload	1	0.1
Step 1.DETERMINE item for unloading	1	0.1
Step 2.RETAIN selected item for unloading & VERIFY goal satisfaction status	2	0.2
Step 3.RETURN with goal accomplishment	1	0.1
Method for goal: MOVE to item to unload	1	0.1
Step 1.RECALL selected item for unloading	1	0.1
Step 2.DETERMINE - position of selected item	1	0.1
Step 3.ISSUE robot command to move to position of selected item	1	0.1
Step 4.WAIT for response of robot	1	100.1
Step 5.VERIFY goal satisfaction status & RETURN with goal accomplishment	2	0.2
Method for goal: PICK item to unload	1	0.1
Step 1.ISSUE robot command to pick-up item at location	1	0.1
Step 2.WAIT for response of robot	1	24.6
Step 3.VERIFY item has been picked (goal status)	1	0.1
Step 4.RETURN with goal accomplishment	1	0.1
Method for goal: Move to EXIT unloading area for item	1	0.1
Step 1.RECALL entry point to the area	1	0.1
Step 2.ISSUE robot command to move to entry and exit area	1	0.1
Step 3.WAIT for response of robot	1	162.4
Step 4.VERIFY goal satisfaction status & RETURN with goal accomplishment	2	0.2
TOTAL	45	457.1

The method "Move to block of unloading area for item" (table 4-4) has six primitive operators. The first operator "Recall unloading area for item" retrieves the unloading area. The following three operators constitute the main role of the behavior codelet: to determine the unloading block area, to issue high-level command to the robot – move to the unloading area, and to wait for a response or command feedback from the robot. The fourth operator is used to verify and evaluate the effects of the behavioral action by the codelet, and this is done by an associated expectation codelet. The evaluation of an expectation codelet may follow a perception that in turn follows the behavioral action. The issuing of a command and waiting for robot response call for communication of the robot controller and the robot simulator system over the network.

4.7.2.3 Complete NGOMSL Analysis Using QGOMS Tool

The QGOMS (Quick and dirty GOMS) tool provides easy understanding of tasks and sufficient analytical accuracy (Beard et al., 1996). The main advantages of the QGOMS tool are: (i) the conceptual complexity of GOMS is reduced since learning the subtle differences between goals, methods, and operators is not required; only task nodes are used, (ii) using the tree notation and direct manipulation allows faster GOMS model development; less hand motion and typing to enter the model electronically, And (iii) it automatically computes the task-completion times and learning times, allowing faster model construction and more time for "what if" analysis.

To make a GOMS analysis, we need to setup a specific world environment and a fully defined task outline. Suppose the warehouse floor is configured to have one type-1 item in the unloading area, a charger at one of the charge stations, and the robot at the center of the floor. Also, suppose there is an order of type-1 item that to be processed at shipping area 1. Since GOMS analysis assumes a predetermined order of task execution, the robot executes the following tasks in that order.

1. Unload and shelve type-1 item.

2. Process order 1, which needs a type-1 item.

3. Charge battery.

After entering the task information into the QGOMS tool, we get the full analysis output, which gives the total average execution time to be 2020 seconds and the total learning time to be 5400 seconds. The complete QGOMS analysis output is given in Appendix D. Table 4-5 summarizes the analysis for the subtasks: "unload item", "shelve item", "unshelve item for order", "ship item for order" and "charge robot." The average, minimum, and maximum execution times for external actions are decided from statistical average of human experiments that are performed on the same simulated robot environment.

Table 4-5. Summary of QGOMS analysis (Appendix D) for the example task.

Task Method	Execution time (sec)			Learning time (Sec)
	Min	Average	Max	
Unload item	216	457	700	1110
Shelve item	234	429	594	1110
Unshelve item for order	240	443	618	1110
Ship item for order	222	452	681	1110
Charge robot	77.7	239	435	930
Total	989.9	2020.2	3028.2	5400

Let's briefly discuss how the execution times for the GOMS analysis are compiled from human experiments. Human operator, a friend of mine who does not have any knowledge of our computational model, performs behavioral tasks with external action (with "wait for response of robot") multiple times. Such is the "Move to item to unload" behavior that moves the robot from the entry of the unloading area to near to the selected item to be unloaded. Using a graphics interface program to the robot simulator environment, a human operator could press a combination of keyboard keys to manipulate the robot. For instance pressing "a –d" causes the arm to move down. Our objective here is to measure the time required to complete the external action once the command to perform a behavioral task is invoked. Pressing a combination of keys is considered as an automatic process to invoke an external command and the time taken by this process is not integrated as part of the execution time for the external action. The time taken by the mentioned behavior depends on the distance from the entry point to the item position.

There are three possible positions for items in the unloading areas and each of these item positions has different distance from the entry. At a specific run of the external action of the behavior, the robot moves to the item covering the associated distance. Assuming the robot moves at a constant speed, the execution time for the external action is linearly proportional to the covered distance. From all the trials, the minimum (43.2 sec.), average (100.1 sec.) and maximum (144 sec.) execution times for the behavior are computed. Such triple execution time representation is accommodated in the QGOMS task analysis tool. With similar human trials the minimum, average and maximum execution times of other behaviors with external action are computed. The execution time from a particular run of a behavior with external action may have an execution time close to the corresponding average execution time but should be with in the range between the corresponding minimum and maximum execution times. The QGOMS task analysis shown in appendix D is constructed using execution times obtained from such human trials.

4.7.2.4 Comparing the Agents Run with NGOMSL Analysis

The robot has an initial charge of 150,000 energy units and the warning bell to recharge sounds when its charge level is below 45,000 energy units. The robot controller decision making mechanism (behavior net) is tuned to follow the same task priority list used in the QGOMS analysis above. In general, when multiple tasks (behavior streams) are executable at the same time, they compete to control the agent. Such behavior streams (tasks) can interleave each other. NGOMSL/QGOMS analysis assumes that tasks are performed serially. Therefore, if the analysis gives importance to the parallel execution of tasks and the intra-task interleaving/interruption, then such tasks are not candidates for NGOMSL/QGOMS analysis. All the domain tasks are serial and only one behavior could be active or have control at a time. Therefore, parallel execution at the behavior or method level is not an issue. Interleaving of tasks or behavior steams could happen, but it is not an important issue in our GOMS analysis. Moreover, tuning the behavior net gets a predetermined task prioritization or task execution order, and this satisfies the requirement for NGOMSL/QGOMS analysis. The objective of our test here is to compare the run of the robotic agent to the QGOMS analysis and find out if the two approximate each other with a specified operator sequence and the evaluation measure being the estimated execution time. This quantitative measure allows us to evaluate how well the robotic agent executes the tasks as expected or predicted by the GOMS analysis.

Table 4-6. Comparisons of predicted and robot execution times for tasks in the warehouse domain.

Task/subtask	**Execution Time (sec)**	
	Predicted average	**Robot run**
Unload item	457	476
Shelve item	429	397
Unshelve item for order	443	362
Ship item for order	452	420
Charge robot	239	383
Total	2020	2038

The behavior net control parameters, the robot state and the environment were setup so that the given subtasks (unload, shelve, unshelve, ship and charge) and their underlying sequence of operators could be performed by the robot in the predetermined order. The objective the NGOMSL analysis is to compare the performance of the behavior net and a human operator in controlling the robot operations provided that both execute the robotic tasks in exactly the same order. As shown in table 4-5, this task, as per our QGOMS analysis, is predicted to have an execution time: minimum of 990 seconds, average of 2020 seconds, and maximum of 3028 seconds. The actual robot run that perform the same task in the same order takes an execution time of 2038 seconds which is very close to the expected average. Such closeness of the actual and predicted execution times of the aggregate task does not imply the same degree of closeness in comparing the actual and predicted execution times of each subtask. But the actual execution times always fall between the predicted minimum and maximum execution times.

4.7.3 Experiments to Control Priority of Goal Context Hierarchies

One of the additions we have over Maes' mechanism (1989) is that we introduce the drive concept as the primary goal directed motivation to goal context hierarchies. Each drive has an importance parameter. Based on the value of its importance parameter, a drive modulates the amount of goal-end motivation energy that it passes to competencies that are directly hooked to it. This is unlike Maes' mechanism in which the same level of such motivation passes to all connected competences. The other addition is the control of the strength of activations passing among connected behavior streams (goal context hierarchies) using a global parameter we call discrimination-factor. In this section, we will show how the importance and discrimination-factor parameters are used to tune a strongly goal directed behavior net to control priorities of competing behavior streams.

4.7.3.1 Experiment Setup

In this experiment, we setup a warehouse configuration that allows the availability of competing tasks that the robot can deal with. The other constraint for the experiment is to fix the priority for each task so that the robot could be performing the tasks in that order.

One such configuration is to have three type-1 and three type-2 items in their corresponding unloading areas; the robot starts at the center of the warehouse and has initial charge energy of 300,000 units. Also, there is no need to unshelve and ship items. At the end, the robot should move all items to be in their designated shelving areas. For our experiment, the order of execution of tasks is set as follow: (i) charge robot as soon as the energy meter shows below the warning level, (ii) unload and shelve all type-2 items, and (iii) unload and shelve all type-1 items.

The behavior net global parameters are set as follow: motivation from drives (γ) is 500, motivation from state (ϕ) is 0, influence from protected goals (δ) is 50, mean activation level (π) is 20, threshold (θ) is 50, and discriminative factor is 0.99. These parameter settings make the behavior net completely goal-directed, more adaptive or sensitive to external motivation and less biased to internal activation history. The importance parameter values are set as follow: (i) 0.99 for charging drive, (ii) 0.85 for type-2 item unloading/shelving drives, and (iii) 0.15 for type-1 item unloading/shelving drives. The discrimination factor parameter value is set to 0.99.

We have three set of robot runs. First (IDA-1) is using a behavior net controller tuned by the above parameters setting. The second (IDA-2) is the same as IDA-1 experiment without the effect of the discrimination factor parameter. The last (MAES-1) is with Maes' equivalent goal-directed behavior net instance that is tuned by the above parameters setting but without the effect of importance and discriminative factor parameters. In the experimental setup, the "unloading type-1 item" behavior stream competes with "unloading type-2 item" behavior stream and as long as there is an item of each type in the unloading areas.

We will analyze the effects of importance and discrimination-factor parameters by comparing the activation or motivation level of behaviors from each behavior streams. To ensure a better accuracy of the experiment, we need to keep all other variables that affect the dynamics of the behavior net constant. Besides the parameters, the dynamics in the behavior net is dependent on the structural features of constituting behavior streams. Such features include depth/breadth of behavior streams, the different relations (connections) among competencies, and fan-in/fan-out at each competency module. The two behavior streams that we consider have similar structures and therefore the structural features are safely assumed to be invariant. Thus our tests will reflect the effect of changes on values of the importance and discrimination-factor parameters.

4.7.3.2 Result

As expected, deterministic prioritization control of behavior streams can not be achieved in the MAES-1 experiment. As such, (i) items are unloaded and shelved in any order, (ii) the charging behavior stream could not be guaranteed to execute before robot runs out of energy. But, the IDA-1 experiment shows the robot performing the tasks in the required order with 100% accuracy; that is, (i) type-2 items were unloaded and shelved completely before type-1 items were unloaded and shelved, (ii) Charging task starts and completes by preempting the items unloading/shelving tasks, and (iii) in doing so, a highest priority executing behavior stream is not interrupted by or interleaved with that of a lower priority executing behavior stream.

The Maes' mechanism can set priority only by building causal orders among competence modules. But, this is not an elegant way of controlling priorities: (i) the complete behavior net may need to be reprogrammed when ever a new behavior stream is integrated, (ii) priorities of behavior streams (tasks) could not be changed during run-time, and (iii) parametric control of task-level motivation is more plausible cognitive mechanism than explicit programming of priorities using causal links or rules.

To keep priorities of tasks, the corresponding behavior streams should have executable behaviors with motivation level reflective of the set priority. An executable behavior that controls the robot's action is the one with the highest motivation among the executable ones. In general, maintenance of priorities is possible if and only if, considering all executable behavior streams with specified priorities, the winner behavior should be under the one with the highest priority. Next we will analyze how importance and discrimination-factor parameters contribute to the motivation level dynamics to control priorities.

4.7.3.3 Analysis

Effect of Importance Parameter

The value of importance parameter associated with a drive determines the goal end motivation level of behavior streams connected to the drive. All other things being equal between two behavior streams, the one motivated by a high importance value is expected to have a higher average activation level than the other motivated by a low importance value.

Figure 4-14 shows the linear plot of the activation level pattern of two behaviors (Move-to-Unloading-Entry-1 (MUE1) and Move-to-Unloading-Entry-2 (MUE2)) that performed under MAES-1 experiment (effects of importance and discrimination factor parameters are disabled). MUE1 and MEU2 behaviors are correspondingly under the competing "Unload type-1 item" and "Unload type-2 item" behavior streams.

In this and similar plots that follow, the y-axis shows the activation level and the x-value is time. The data is obtained by taking snapshot of activation/motivation level of behaviors at strategic intervals as the instantiated behavior streams are involved in the behavior net dynamics while the robot perform the tasks. The plots are not exact activation patterns but is representative of the general trend that we want to show.

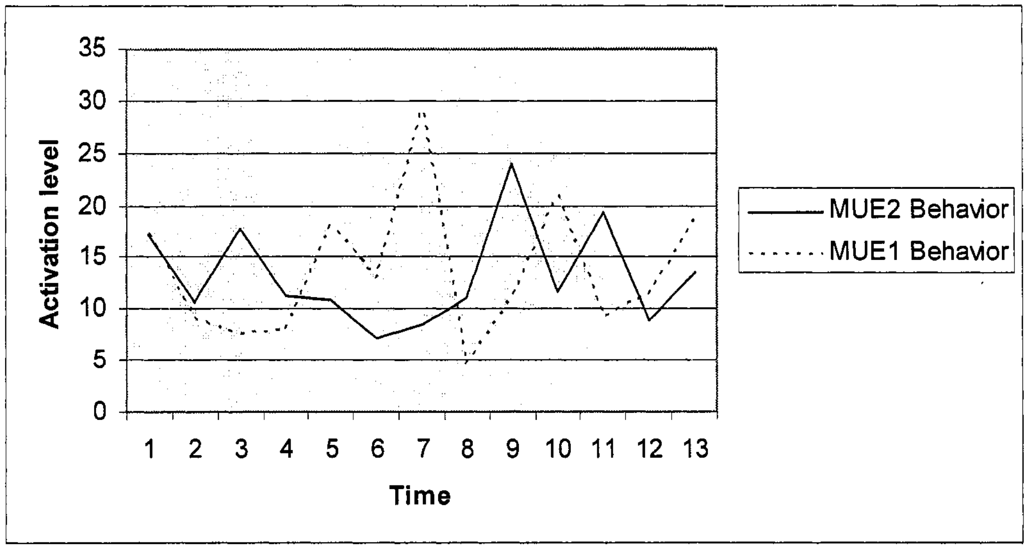

Figure 4-14: Variation of motivation level of competing behaviors in Maes' mechanism (MAES-1 Experiment).

As we discussed above and as the plot (figure 4-14) shows, Maes' mechanism does not allow control of priority – there is a single goal-end motivation level for all behavior streams.

Figure 4-15 depicts a plot of activation-level the MUE1 and MUE2 behaviors from a behavior net dynamics with the use of variable importance settings to different drives and but without the effect of discrimination factor (IDA-2 experiment). The plot shows that importance parameters help to set and maintain motivation of behavior streams at required relative levels. The MUE2 behavior (under a higher importance behavior stream) maintains a higher activation level than the MUE1 behavior (under a lower importance behavior stream.

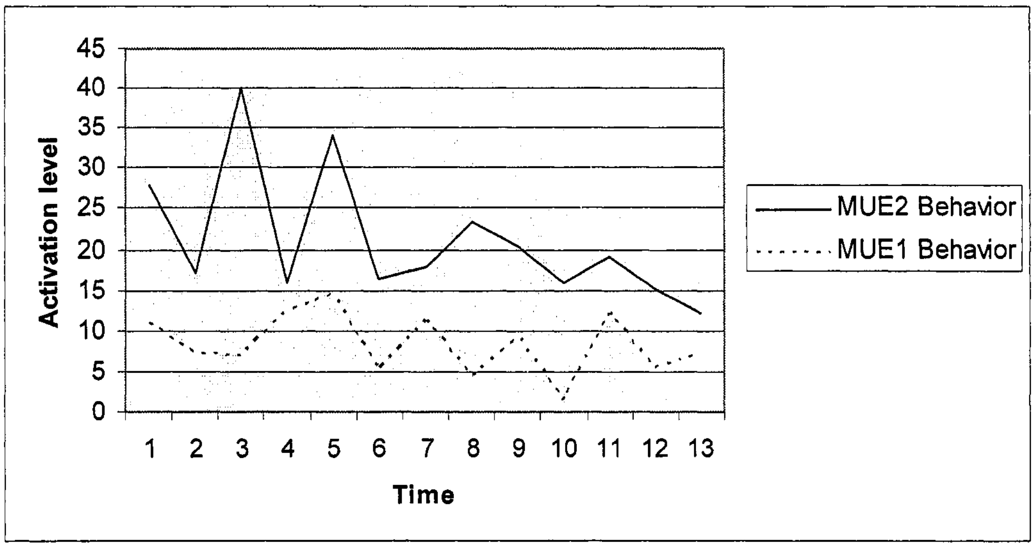

Figure 4-15: Variation of motivation level of competing behaviors with the effect of importance parameter (IDA-2 Experiment).

But, a behavior with a given importance could have a lower motivation level than a behavior with less importance or vice versa. This is mainly due to internal passing of activation among interacting behavior streams The net effect of the intra behavior stream activation passing over time could be to increase the motivation of a low-importance behavior stream at the expense of the high-importance behavior stream. As a result, priority of tasks could be violated since a behavior under a low-importance behavior stream could eventually get higher activation level than a behavior under a high-importance behavior stream. This fact is evidenced by the plot of figure 4-16 (from IDA-2 Experiment) that shows activation level of two directly interacting behaviors: Move-to-Unloading-Block-1 (MUB1) and Move-to-Unloading-Block-2 (MUB2), which are correspondingly under “Unload type-1 item” and “Unload type-2 item” behavior streams.

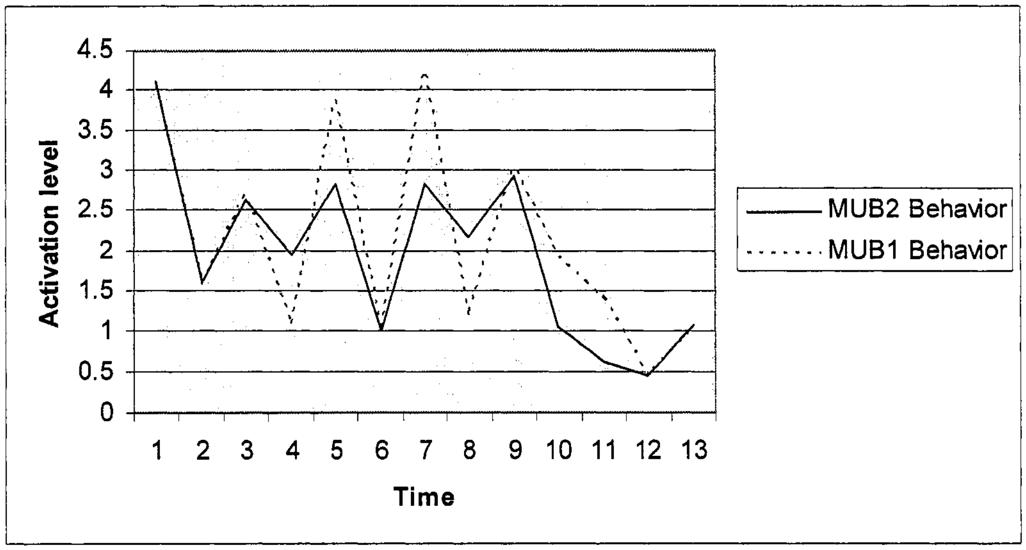

Figure 4-16: Variation of motivation level of competing behaviors. Importance parameter is not enough to set priority (IDA-2 Experiment).

Effect of Discrimination-Factor Parameter

The discrimination-factor parameter is intended to complement the effect of importance parameters by fixing the short coming shown in figure 4-16 in controlling priority of tasks. This parameter, by weighing the intra behavior streams activation passing, modulates the independence of a goal context hierarchy from the effects of other competing/cooperating goal context hierarchies. Considering a high importance goal context hierarchy, we conjecture that a *high discrimination-factor parameter value determines the concentration level in performing the associated task and vice versa.*

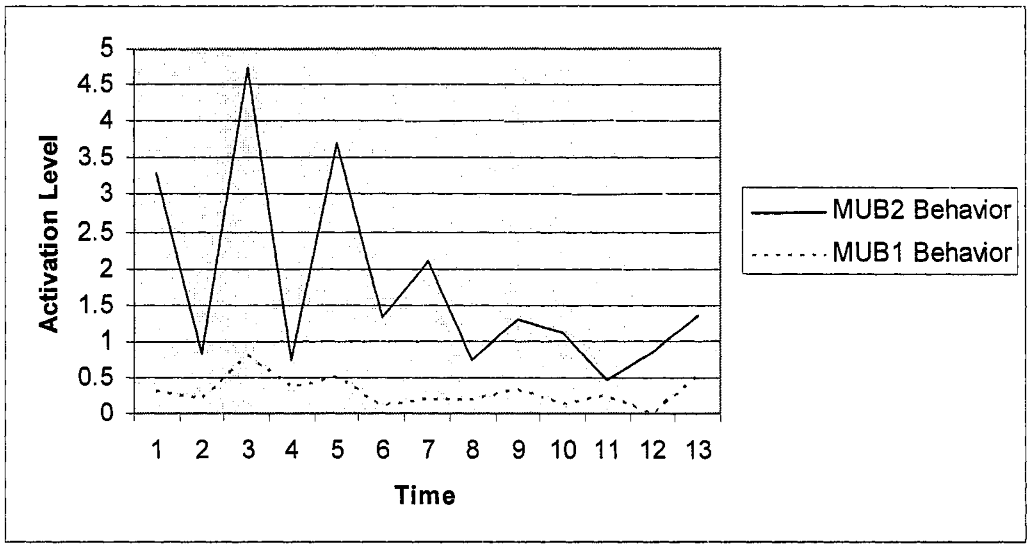

Figure 4-17: Variation of motivation level of competing behaviors with effect of importance and discrimination-factor parameters (IDA-1 Experiment).

Figure 4-17 plot (IDA-1 experiment) shows that MUB2 behavior with high importance has consistent higher motivation than MUB1 behavior with low importance. The improvement in priority controlling (compare plots in figures 4-16 to figure 4-17) is obtained by the effect of discrimination-factor parameter. Thus, the addition of the importance and discrimination-factor parameters in our behavior net mechanism allows control of priorities to goal context hierarchies.

As the rest of the behavior net parameters, there is no general method of tuning these two parameters. But, we argue that priority could be set in a highly goal oriented, less opportunistic, and with stronger bias or sensitivity to external goal-end motivation. In our robot environment, priority of tasks could be changed at run time by controlling these parameters. Such control may need a type of meta-goal-context system, which is beyond our research objective here. IDA's conceptual model proposes to influence decision making using emotions and feelings; influencing the behavior net is possible by tuning the control parameters (of the behavior net) towards required features such as the relative priority of behavior streams.

4.7.4 Compared to Human Operator

The QGOMS analysis we discussed above provides predictions of the time needed by the robot to execute the different tasks in the warehouse domain. But, this analysis is based on the average execution time of different runs of behaviors that invoke external actions under different environmental configurations. In this sub section we will compare the execution times of the robot and a human operator in executing a task in the same order and under the same environmental configuration. A male human operator, a nurse by profession, is told how to operate the robot in the simulator environment using a graphics interface. After a short practice of using the interface, he is instructed to operate the robot to perform the task. He is instructed to control the movement of the robot in an efficient way (short route) he thinks provided that he does not violate any constraints he may need to adhere in the movement. He is instructed that he can take as much time as he wants to make his decision but he needs to think about the quality of his decision so that he will not loose efficiency. For the purpose of our comparison, we will ignore the time taken by internal actions – execution time of internal actions are very short compared to the external actions and also we assume that the robot and the human may employ approximately the same type and number of internal actions. The robot does not have the complete model of its world; it does not sense or does not have knowledge about everything in its environment. For instance, the robot knows about the entry point to a processing area – at center of the open end and a little passed the boundary of the area. But it does not know how wide the opening is or the end points to the opening. When performing tasks, the robot moves from one processing area to a block or from a block to a processing are or from a block to another block using built-in knowledge of few points in the warehouse as landmarks.

For this test, we use a simple task of unloading, shelving, unshelving, and shipping an item and then charging. Shelving and shipping of item is done at designated positions. The human operator and the robot do the same task under the same environmental setup. The human performs the task under two different knowledge levels. These are, HO-1 - using the same knowledge of navigation points the robot has, and HO-2 - using human operator's own decision faculty to freely navigate during the execution of the task. Behavior streams, behaviors, and behavior codelets are designed and implemented so that they can produce actions as per specified by the designer. The comparison of the runs of the robot and HO-1 allows testing of how well codelets, behaviors and behavior streams as integrated entities perform their designed role; the robot is judged to be effective if it exhibits efficiency comparable to a human that is constrained to operate using the same navigational domain knowledge as the robot. As figure 4-18 shows, the execution times to perform the complete task and the subtasks of the robot and HO-1 are very close. To complete the task the robot took 2038.2 seconds while the HO-1 took 1987.2 seconds and we can conclude that for the given domain knowledge and environmental constraints, the designed behavior streams perform their designated tasks very well.

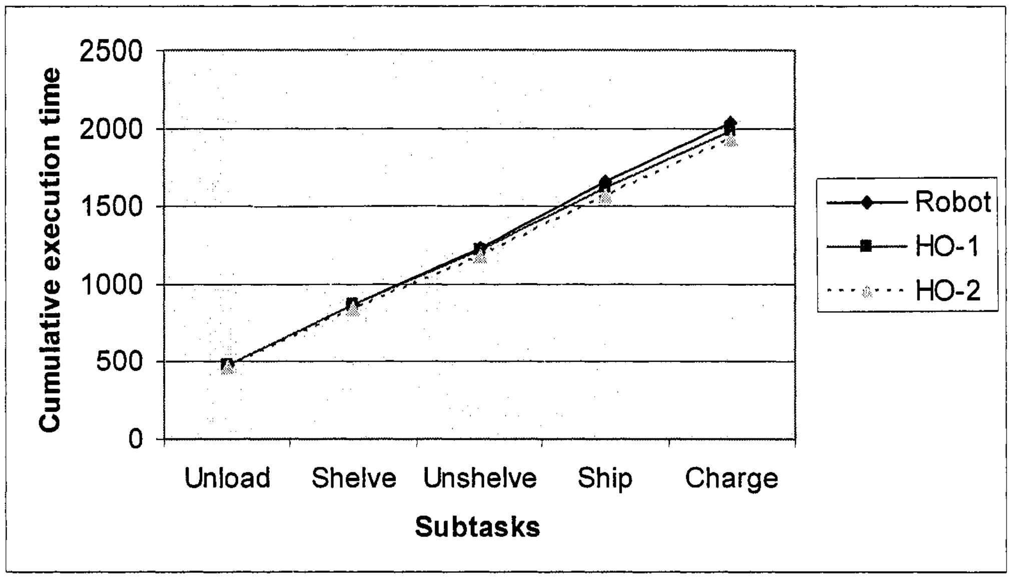

Figure 4-18: Comparison of cumulative execution times of a human operator (HO-1, HO-2) in performing a given task with unload, shelve, unshelve, ship, and charge subtasks.

If not impossible, it will be hard to incorporate a complete world model and a rich perceptual mechanism to a software agent system. All knowledge is built-in unless the agent has a capability for learning. An agent with limited world knowledge and perceptual mechanism is designed and implemented hoping that it will perform tasks well enough to replace human operators. The comparison of the robot and the HO-2 runs is to test how well the robot controller is designed to approximate human performance in executing domain tasks. Our test shows (figure 4-18) that the human operator completes each subtask and the full task (1926.5 seconds) faster than the robot. This is expected since a human operator has richer in knowledge. Human operator could use more efficient routes of movement. Human operator could also use new methods that the robot does not have; these methods could be efficient to particular or exceptional situations that arise on some instances of tasks. Our conclusion is that for this particular and relatively simple domain, the behavior net as a robot controller could perform well. But, we do not claim that the behavior net will reproduce such good performance in more sophisticated environments.

Besides our comparison with human operator, for a couple of reasons, we did not make additional tests of our behavior net mechanism with other action selection mechanisms. First, our domain task and environment setup are unique and we could not find readily available research data that we can compare to our test results shown above. Second, Maes' mechanism is very similar to the instantiated behavior net of IDA and Tyrell (1993) has compared Maes' mechanism with other well known action selection mechanisms such as subsumption (Brooks, 1986, 1990), fine-grained subsumption (Rosenblatt & Payton, 1989) with extensive detail and we do not see a need to repeat one here. But, these and other related mechanisms are discussed briefly in the next section.

4.8 Related Works

AI has two approaches for the action selection problem – those of classical planner and behavior-based (situated AI) mechanisms. A reasoning process called planning, using an agent's internal representation for goals, actions, and events, produces a sequence of actions that can achieve a given goal in a given context (Georgeff, 1987). While planning systems can handle a certain class of problems well, they have drawbacks such as brittleness and slow reaction time, particularly in noisy and dynamic environments. In general, plans assume that the world will not change in unexpected ways. Also, building plans take time, and planning systems are not well suited for reactive systems such as robots and simple autonomous creatures.

A number of researchers (Brooks, 1986; Maes, 1990; Minsky, 1986; Rosenblatt & Payton, 1989), motivated by work of ethologists, proposed behavior-based architectures with the central thesis that control systems for situated agents or robots should be decomposed according to desired external behaviors of the system, rather than divided along functional lines based on the structure of internal mechanisms (Rosenblatt & Payton, 1989). For simple reactive systems, the forwarded argument is that distributed configuration of individually simple behaviors, directly interacting to the sensors and actuators of an agent, could perform better than planning systems, particularly in dynamic, noisy, uncertain environments, and where real time performance is a requirement. Behavior-based mechanisms do not explicitly plan; they respond to changes promptly since they re-evaluate their environment and modify their course of action with each step of time. They are more robust and opportunistic, less optimal and deliberative. Behavior-based systems could be goal-driven and situation-driven. Behavior based mechanisms try to address how simple creatures can produces relevant behaviors in their environments; they do not try to explain how people may handle planning problems. Below, we briefly review some of the well-known behavior-based mechanisms. One such is Maes' (1989) behavior network, which is already discussed in chapter 2.

4.8.1 Subsumption Architecture

Brooks (1986) argued that horizontal decomposition of control, in the order of perception, world modeling, plan generation, and plan execution, creates bottlenecks, as all sensor data must be fused and all tasks must be coordinated before any actuator is activated. The subsumption architecture addresses the problems of horizontal decompositions by arranging behaviors as levels of competence in vertical decomposition (see Figure 4-13a). Each behavior is made up of a collection of finite state machines and encapsulates the perception, planning and task execution capabilities for a narrow aspect of agent control. The structure can be partitioned at any level, and the levels below are intact with complete and self-contained specification of the agent. To provide greater functionality, competences can be built at higher levels over the competences of the levels of the layered architecture. A behavior module is implemented as an augmented finite state machine with internal variables as well as states. Each module receives messages via multiple input lines and generates messages on multiple output lines (see Figure 4-13b). The critical feature of the subsumption architecture is that control is distributed among behaviors that operate asynchronously and in parallel. Intelligent control emerges from the interaction of behaviors via the message carrying lines. Behaviors at the higher levels subsume (at input lines) and inhibit (at output lines) behaviors at the lower levels. Both the subsuming and inhibiting control lines block messages for a specified period of time. The subsuming control also implies that the higher level behaviors can inject their own command message on the input lines of lower level behaviors.

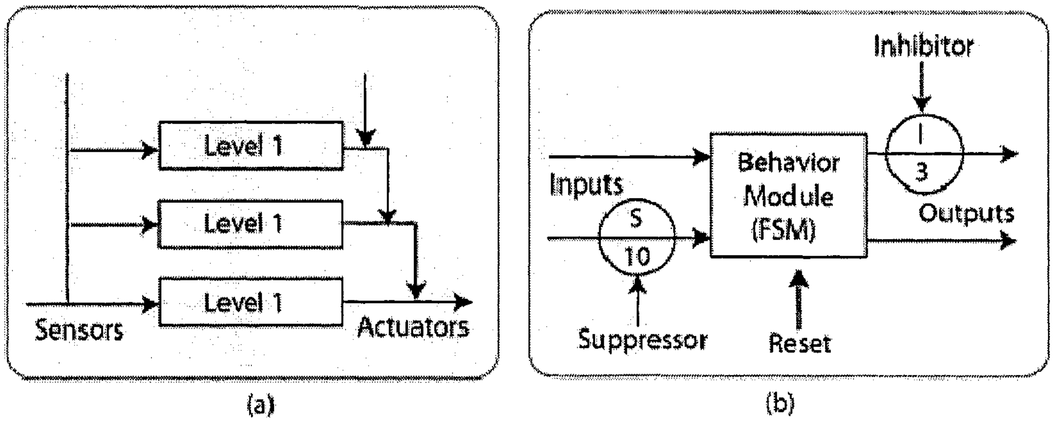

Figure 4-19: (a) Vertical decomposition of control layers of subsumption architecture; (b) Subsumption behavior module – a finite state machine: (i) receives input data that can be subsumed, (ii) has state that can be reset, (iii) outputs data that can be inhibited, (iv) subsumption and inhibition stay for the predetermined amount of time as shown in the circles (redrawn from Brooks 1986).

The subsumption architecture provides a distributed world representation, in the sense that each behavior module receives input data that is directly relevant to their particular processing needs, and so each layer has a partial view of the world – there is no coherent model for sensory fusion.

Although it has many important features, the subsumption architecture has limitations (Rosenblatt & Payton, 1989). A behavior directly influences another behavior, only by completely subsuming and inhibiting it. In case of conflicts, command fusion between levels is not possible. A semblance of command fusion is possible by tuning timings for subsumption/inhibition and for message transmission rates. But this approach may not be practical with the layered modularity where subsumption/inhibition could come from multiple modules with varying concerns.

When one behavior is inhibited by another, the information content of the inhibited behavior is lost completely, and therefore the concern of the inhibited one is disregarded. With its loss, subsumed behaviors will not be able to adequately perform the functions for which their level of competency was designed. Or, with the addition of higher levels, the lower levels may need to be redesigned, losing the advantage of modularity.

The limitations of the subsumption architecture can be summarized as: (Rosenblatt & Payton, 1989): (a) inadequate command fusion, (b) inaccessibility of internal state, and (c) disruption of level of competence. Rosenblatt and Payton (1989) developed a fine-grained subsumption architecture that overcomes these limitations.

4.8.2 Fine-grained Subsumption Architecture

Roboticists Rosenblatt and Payton proposed a connectionist architecture that provides a solution to the problems identified in the subsumption architecture by making behaviors as fine-grained as possible so that a competence module is without inaccessible internal states. This model uses the same vertical layering as subsumption architecture. However, behaviors are not self-contained finite state machines, but rather from networks of atomic functional elements that are without internal state and instance variables. Each unit, representing an indivisible specific concept that a designer desires to establish, receives multiple weighted inputs from other units and from external data sources, computes its activation level and generates a single output (Rumelhart & McClelland, 1986). The node is a fairly standard artificial neuron, except that the node activation can be *any* function of the weighted inputs, not necessarily just a weighted sum. The activation and output functions for each unit can be any mapping from real numbers to a single real number value. The only constraint is that these functions be defined for inputs from -1 to +1 inclusive, and that the outputs be within the same range. There are no constraints on how the units are interconnected, except by a desire to maintain structured behavior-based subsumption architecture.

Since units are without internal variables, they are completely transparent, and there is no inaccessible internal state. Also, each behavior is distributed among several units so that the desirability of choices can be expressed by the output value of the corresponding units (Rosenblatt & Payton, 1989). Negative outputs indicate undesired choices, and positive outputs indicate desired choices. The magnitudes of output values indicate the degree of desire/undesired choices. Each behavior expresses its weighted preferences to command units or motor actions. The preferences of multiple behaviors are combined at command units, and the action taken is the one that can best satisfy the constraints of multiple independent behaviors simultaneously. So, command fusion is possible by allowing nodes of one behavior directly communicating with the nodes of other behaviors – a distributed command arbitration process. In building new levels in the layered architecture, new behaviors do not completely subsume the function of existing behaviors, but simply bias choices in favor of various alternatives - a possible weak sense of subsumption. Hence, established levels of competences remain intact and also can forward their concerns independent of the concerns of competences at higher levels. Without complete subsumption, it is unlikely that higher-level behaviors cause disruptions at lower-level behaviors.

Networks of such simple units (without internal variables) can realize coherent and sophisticated behaviors using the "selected unit update" mechanism. When a new value is entered into an input unit, the links emanating from that unit not only carry a numerical value, but also transmit a tag that is unique at a one time step. The receiving units use the tags to conditionally update their activation level and output. This simple marker propagation mechanism allows 'latching" of nodes, so that a network constitutes a finite-state machine type of predictable behavior that can maintain the integrity of its decision. There is a free flow of information since no information is hidden in units and there is no communication barrier among behaviors.

In this architecture, behaviors are linked into a network; preferences of a behavior are combined to the preferences of other behaviors; and a command with the greatest weight wins. As Blumberg (1996) points out, the main problem with this approach is that the weight used by a given behavior is for a given action and cannot be set independently of those used by other behaviors. That is, weights have to be carefully chosen and the introduction of a new behavior may necessitate the re-tuning of the weights. Tyrrell (1993) also argued that care needs to be taken with respect to how the weights are combined, so that important behaviors are not overwhelmed by the peripheral desires of many other behaviors.

4.8.3 Free-Flow Hierarchy Architecture

Tyrell (1993), based on the fine-grained subsumption architecture (see section 4.8.2), proposed an ethologically inspired behavior-based action selection model called a free-flow hierarchy. This architecture adapts a hierarchical approach (Tinbergen, 1950) in which only drive-based activation is injected at the top of the hierarchy – providing a different sense of hierarchy from that of subsumption architectures. Unlike fine-grained subsumption architecture, the free-flow hierarchy does not limit the activation level within the range between -1.0 and 1.0, nor does it explicitly use the "selective unit update" mechanism. In free-flow hierarchy, all nodes in the hierarchy can influence the subsequent behavior of the agent. The architecture does not restrict the number of behaviors that remain active at any moment. Open competition is allowed, and excitation/inhibition is inhibited across systems (behavior streams for a specific task). Behaviors express weighted preferences for behaviors lower in the hierarchy. The process of propagating preferences covers the entire hierarchy, but decisions are always deferred to the lowest (i.e., action or command) level, where the most highly preferred action (based on a winner-take-all decision scheme) is chosen.

The free-flow hierarchy architecture performed well in Tyrrell's simulated environment, but there are serious concerns in its applicability for complex domains. As Blumberg (1994) pointed out, in fine-grained subsumption and free-flow hierarchy architectures, the ability to arrive at a compromise decision comes at the expense of complexity, in which a behavior expresses a weighted preference for a given action. Weights need to be adjusted carefully and adding a new behavior may force tuning existing weights. The action selections made at the lowest level come with inefficiency as a result of extensive checks and balances in combining preferences. Complexity for decision is high, particularly if the lowest level has a large number of nodes along with extensive processing that is associated with the calculation of global sensory information and preferences from all behaviors. Inefficiency is incurred by neglecting to provide a focus of attention since sensory data is processed in all levels of the free-flow hierarchy. Thus, this architecture may not be scalable or applicable for real-time solutions. Drives constitute the high-level nodes in the hierarchy. Behaviors usually operate in service of multiple drives, but this cannot be supported in free-flow architecture. In general, all hierarchical structures are inherently inflexible in the sense that connections between nodes can not be easily altered.

4.8.4 Inhibition and Fatigue Architecture

Blumberg (1994) proposed an action selection architecture based on the ethologically-inspired model first presented by Ludlow (1976, 1980). In Ludlow's model, an activity has a value that is based on the sum of its relevant internal/external factors minus the inhibition it receives from competing activities. Activity "i" inhibits a competing activity "j" by an amount equal to the product of activity i's value and an inhibitory gain K_{ji}. Ludlow pointed out that if (1) activities are mutually inhibiting, (2) the inhibitory gains are restricted to a value greater than 1, and (3) values of activities are restricted to zero or greater, then the model would result in a winner-take-all system. That is, an activity with a non-zero value is said to be active, and in a stabilized system, only one activity will have a non-zero value.

4.8.4.1 Greater Control over the Temporal Aspects of Behavior

Action selection mechanisms should solve the problem of providing the right amount of persistence. Blumberg (1994) notes the importance of controlling the temporal aspects of behavior to attain the right balance between too little and too much of persistence. Too little results in a dithering among activities, and too many results in the loss of opportunities by mindlessly perusing a goal to the detriment of other goals. The Bloomberg/Ludlow approach uses the inhibition and fatigue mechanisms to control persistence.

Inhibitory gain factors set the level of persistence for each activity. The higher the inhibitory gain, the stronger the persistence level of the associated activity. By modifying inhibitory gains, the relative level of persistence compared to competing activities could be adjusted.

To give a better chance of being active to low priority activities in the presence of high priority activities, a fatigue mechanism associated with every activity. When an activity is active, its fatigue level increases in proportion to the activity's value. Over time, fatigue reduces the value of the active activity. When the activity is no more active, the fatigue level decays to zero, and the value of the activity starts to build. The fatigue mechanism models a time-sharing system that reduces the chances of persistence towards a given goal at the expense of others.

Each behavior assesses its own relevance in its given internal and external factors. Behaviors rely on Releasing Mechanisms to filter their sensory input so as to identify significant objects and events whose presence helps to determine the current appropriateness of the behavior. Releasing Mechanisms output continuous values that represent the strength of associated sensory input. Internal states such as hunger are represented by endogenous variables that output a continuous value. A behavior computes its relevance or value by combining the values of associated Releasing Mechanisms and endogenous variables – an expression of internal and external factors using a common currency. At every step of action selection, a Releasing Mechanism updates its value, based on the presence or absence of relevant stimuli. Parameters such as range values of Releasing Mechanisms can be adjusted to make a behavior more opportunistic. As a hungry creature, searching for food may end up near a pond, this may cause its Releasing Mechanism associated with water to output a high relevance value for the behavior of "quench-thirst" and for the creature to exploit the opportunity afforded by the environment.

4.8.4.2 Loose Hierarchical Structure with Information Sharing

Blumberg's model is implemented as a winner-take-all hierarchical system with the ability to share information among different behaviors. At any one time, only a single behavior is a winner to be active and to control actions. But behaviors that lost the competition can post suggestions in support or against various low-level actions. The winner has access to posted suggestions (information sharing) and considers them in formulating the behavioral actions. Bloomberg stress on the strength of his model by pointing out that it exploits the important features of hierarchical structures to represent complex behaviors; the winner-take-all arbitration approach, grounded on ethological approaches, provides a focus of attention that avoids processing irrelevant sensory data; and a winning behavior has control over how it wishes to use the recommendation of others.

This architecture is similar to Maes' behavior network (1989, 1990) in the sense that both have situational and internal motivation, spread activation among behaviors, have no centralized control, and use a winner-take-all approach. The disadvantage with Blumberg's model may be related to its complexity. Although it resolved some of the complexity problems associated with the free-flow hierarchy, a huge number of checks and balances still remain among different nodes, and this may limit its scalability. Also, representation associated with each behavior is complex; therefore, in this model, adding a new behavior or a deliberative process like planning may not be an easy task.

4.9 Conclusion

This chapter describes how a goal context hierarchy system, in the framework of the Global Workspace Theory, is realized with features that accommodate subgoaling, competing goals, interleaving of tasks, situation and goal directedness, persistence, and deliberation. Conscious and the unconscious processes compete and cooperate whether the role they play is for deliberation, decision making, perceiving or action execution. Among the unconscious, is a gal context system influences what conscious goal images come to consciousness. We designed and implemented IDA's Action Selection Mechanism (IASM) as a goal context hierarchy system that could interact with IDA's "consciousness" module and modeling the mechanism for their interaction – behavior net influencing what comes to a global broadcast and which in turn affects what behavior is selected as a dominant goal context. The IASM is realized to have this interaction with the "consciousness" module to support decision making at different awareness levels - voluntary goal selection, unconscious stream of actions, and consciously mediated streams of actions. The IASM also allows parametric control that could be adjusted by others mechanisms such as the agents' metacognition module.

The design and implementation of the IASM follows the OOP methodologies. The design and implementation effort includes the separation of the domain independent and domain dependent components in the architecture. While the domain independent component is implemented as java classes, the domain independent knowledge incorporated by the provision of a behavior net knowledge specification language in XML format and inheritable super classes for codelets. The incorporation of the domain knowledge is simplified since: (a) a parser that converts the XML specification to instantiable objects is implemented, and (b) the "how to execute a role" type of domain specific knowledge of codelets can be coded by implementing few abstract methods and with out worrying about the details how and when the methods are used. The discussion includes the use of the IASM development environment as an action selection control system of an agent.

The mechanism was tested as a controller for a ware house robot with competing tasks. As the results and analysis of our tests show, the expected performance was obtained in the robots execution of the competing domain tasks and the control of their priorities.

This chapter also discusses related behavior-based architectures for control of autonomous agents. Unlike the IASM, they are not intended to support deliberative processing. In fact all of them could be a component in the IASM system; for instance the subsumption architecture could be used as a low-level sensori-motor module that gets subsumed by behavior codelets during their action control.

5 Automatization and Deautomatization

Humans have the amazing ability to learn a procedural task so well that they do not need to think about the task consciously in order to accomplish it. This ability is what we call automatization, and allows us to allocate the limited resource of consciousness to cognitive tasks that will benefit most from conscious involvement. In this chapter we describe an automatization and deautomatization mechanism in the action selection system we discussed in chapter 4. Automatization is related to attention and the expected effects of actions. Automatization develops with practice, and with it a shift takes place from serial, attentional, controlled processing to parallel, non-attentional, automatic processing. Once a task has been automatized, there is no need for attention to be paid to its execution unless the expected result does not occur. In our mechanism, such a state is recognized and dealt with in a manner that is cognitively plausible. The deautomatization process detects the failed expectations and temporarily disables the automatization effects. As a result, "conscious" monitoring will once again play a role in performing the action in hopes of discovering and negotiating the previous impasse. Our approach implements the automatization and deautomatization cognitive functions as a self-organizing or an implicit-adaptation system for a conscious software agent.

In the introduction section of this chapter, we will discuss the details of task (procedure) automatization and the underlying system representation components that could be changed by the learning process associated with automatization. In section two, we will discuss the characteristics and definition of automatization. In the following two sections, we will present the automatization and deautomatization mechanisms, which are domain independent automatic processes in the architecture.

5.1 Introduction

We have already mentioned that the overriding task of the mind is to select the next action. In our daily chorus, we execute a multitude of actions. But we have different levels of cognitive control for our actions. Some are voluntary decisions with full conscious awareness. But human actions for the most part are highly automatized and are executed with little or no attention paid to them. From many evidences, we know that automatized or well-rehearsed actions are realized by the interaction of highly coordinated architectural modules (cognitive, sensori-motor, etc.) involving the selection, activation and control of action. The interesting issue for us is how architectural modules or bodily parts learn such a high degree of coordination that makes conscious control unnecessary. Particularly, how do the context hierarchy system and consciousness interact to model automatization in the IDA architecture? To answer this question, we start with the following underlying assumptions for the automatization mechanism.

1. Automatization, as a task-level learning process, develops as tasks are performed over time (or rehearsed) with conscious control or conscious mediation. In general: (i) automatic processes, such as skill acquisition, develop out of *frequent and consistent* experience in an environmental domain (Wegner & Bargh, 1998); (ii) consciousness is a sufficient condition for learning.

2. Automatization shifts the default role of conscious awareness in the execution of procedures (stream of action selections) and improves performance. Conscious awareness fades as automatization develops with experience (e.g., Logan, 1992; Tzelgov 1997; Kanwisher, 2001). Automatization transforms a consciously guided or controlled process into an automatic process.

3. Automatized procedures can be deautomatized, and conscious access can be reintroduced at original points of awareness. Deautomatization usually arises from a failure of actions to produce the expected outcome; it is a mechanism to attract consciousness to faulty or troubled automatic behaviors.

Voluntarily controlled or consciously mediated streams of action selections are transformed into internalized procedural skills that execute automatically and without conscious awareness. Learning happens through experience or when tasks are executed for a sustained length of time. We can think of any human task such as competitive sports, operating machinery (air plane, car, cycle, etc.), walking, cooking, acquiring communication skills (including learning of natural language) and in all of them the degree of involvement of conscious awareness changes with experience. To start with, tasks or new procedures are obtained through different means such as by imitating others, from instructions/explanations (others provide to us), built-in (designed or evolutionary instruction), and creativity. No matter which way we get them, we acquire procedures (or behavior streams or goal context hierarchies) to guide execution to perform tasks. At the beginning, the execution of most tasks starts with the involvement of intensive use of attentional resource. Then experience incrementally reduces and eventually eliminates the attentional control of the executions of tasks. The effect of automatization, if not permanent, stays for long time. But, automatized tasks can be accessible for consciousness via voluntary attention or as a result of a failure to produce the expected outcome by any behavioral action. Deautomatization mechanism removes the automatization effect when a behavioral action fails to produce an expected response.

In this chapter we will try to answer these questions. What is a possible mechanism for learning from experience and what is its effect on the availability of associated events for conscious access? What is a possible mechanism to override the automatization effect in unexpected situations? In the process of answering these questions, we will cover many computational and conceptual issues.

To analyze and implement the adaptations involved in the automatization/deautomatization processes, our discussion in this chapter considers many more factors including: (a) the underlying representation of the system, (b) the learning process, and (c) what sorts of changes the learning process make on the representation. For a conscious agent, we also need to factor the involvement of consciousness in the automatization process and the influence of automatization to conscious awareness.

5.1.1 Task Automatization

A task is a set of stated goals to handle a problem within a given situation, assumptions, and various constraints or requirements. Alternative goal hierarchies or action plans can perform a task. A goal hierarchy performs a task by means of the sequence of decisions for actions it encompasses. The actions of a given goal hierarchy can be automatic, consciously mediated or voluntary. Typically when an action plan is learned for the first time, consciousness plays a major role. As the action plan is executed repeatedly and consistently, experience accumulates and parts of the action plan eventually are automatized. That is, the task is performed with little or no intervention from consciousness. That is, complete automatization is one end of a continuum of cognitive processing in which the opposite end is volitional processing (Whitetaker, 1983). In this continuum, automaticity, a skill acquisition cognitive process, develops with repetition/practice, provided that the task is consistent and predictable. In the volition end of the processing, a task is performed consciously and processing is controlled or strategic. Controlled tasks are characterized as: operating under novel or inconsistent information, with serially accessible limited resources, needing strategic deliberation and an increasing level of attention for reliability where performance is slow. In contrast, automatized tasks are characterized as being: without conscious control or an attended goal image, with wider communication bandwidth for component elements or no serially accessible limited resource constraint, fairly effortless, faster performance, more difficulty to control the process and/or report about. R. de Keyser (2001) gives a detailed overview of automaticity and automatization and their applicability in second language acquisition, such as grammar skills. According to Whitetaker (1983), as automaticity develops, a task can progress from being novel, to variable, to familiar, and then to practiced. Automaticity can be viewed as a gradual fading of conscious involvement.

Automatization helps to save cognitive resources, which are spared for other tasks that require more conscious control.

Starting from infancy, we continuously gain skills and our most developed skills are generally the least conscious in their details. When we first learn new skills, we are thoughtful of each step in the skill process. We are conscious of each step in performing many tasks or behaviors such as driving, walking, and playing instruments before they can be automatized.

In GW theory, attention is the control of access to consciousness. Attention is a process that brings knowledge, skills and perceptual stimuli into consciousness. By consciousness we mean the phenomenal consciousness exhibited when the information (skill, knowledge, stimuli) content of an attention system is broadcast. Attention can be automatic or voluntary. Our assertion here, in agreement to Logan (1992), is that the automatization is a postattentive process and is dependent on attention systems. Once automatized, a task is ballistic after it is initiated by an automatic or voluntary attention. Unless the initiator is a voluntary attention, none of the components of the automatized task, including its initiation, may come to consciousness. The automatization process aggregates units of a task into a larger chunked unit that can perform the whole or a large portion of the task. If inconsistency and unpredictability (or novelty) is introduced, an automated task fails to produce the required outcome. Correcting the failure calls for the involvement of consciousness. Deautomatization reintroduces the service of consciousness to debug the automated task and to make the necessary adaptations or strategic decisions. There is a fairly good body of work on automatization (Shifferen & Schneider 1977; Schnneider & Fisk 1983; Whitetaker 1983; Baars 1988; Anderson 1992; Anderson 1995; R. de Keyser 2001; and others), which provides a framework that shows how practice makes possible the automatization of complex cognitive tasks. In this paper we present a mechanism to automatize a consciously mediated action plan or hierarchical goal context, and a mechanism for deautomatization when the conditions of consistency and predictability are violated.

5.1.2 Dynamic Representation Components Available for Learning

In chapter 4, we have discussed the behavior network system and its integration and interactions with the consciousness module. Particularly, the behavior streams realize the procedures that produce a sequence of action selections for the execution of tasks. We have already discussed that high level constructs in goal context hierarchies, such as behaviors, are underlay by small processing units – codelets. Every thing happens by codelets; the high-level constructs provide representation and control in the higher abstract levels. We have seen that how behaviors may instigate attention, intention and expectation codelets to bring events to consciousness and how, in turn, conscious contents (via broadcasts) instigate behavior codelets to recruit relevant behavior streams as a response to the situation presented in the broadcasts. Below the high-level constructs, the behavior codelets and various types of attention codelets are the low-level processes that realize the interaction of the consciousness and the behavior network system. Any learning mechanism in the behavior network should change some type of representation, which may include: (a) attributes in high-level modules or codelets (like activation), (b) relationships among high-level modules (like links among behaviors), (c) relationships among different codelets (like associations among codelets), and (d) relationships between high-level modules and codelets.

5.1.2.1 Activation Strength and Associations

IDA's architecture has mechanisms that are subsymbolic. The connectionist side allows the computational units of IDA to influence each other using two distinctive processing methods. These are what O'Reilly & Munakata (2000) called activation-based and weight-based processing. In activation-based processing, activation-levels and the spreading of activation/inhibition influence processing. Such processing is used in the action selection dynamics of IDA. In the behavior network, there are high-level modules like behavior nodes that are linked and spread activation/inhibition energy to each other. Also, activation passes vertically from behavior codelets to behavior nodes and vice versa. Weight-based processing is used to adjust associative weight values to influence processing. In the playing field, which implements Jackson's Pandemonium theory (Jackson, 1987), co-active codelets can build associations (links with weight). Two codelets are co-active when they are in action in the playing field in the same window of time. Persistent co-activity between any two codelets will produce a strong associative weight value, and this allows an active codelet to prime the other codelet for action. The priming strength is proportional to the associative weight.

5.1.2.2 What is Changeable?

As in any connectionist information processing system, associated high-level modules and codelets influence each other's processing continuously, based on the strength of association links between them and their activation level. Adaptive processing, or learning, is possible by changing the activation level at high-level modules and at codelets, or by changing the associative weight among connected processing units. The activation patterns emerge from the interaction of processing units where distribution of activity, uniqueness of distribution and temporal stability are representations for higher level cognitive processes.

5.2 Automatization: Background and Definition

In the classical definition, automatic processes, as pointed out by Posner (1978) and Hashner and Zacks (1979) are effortless, unconscious, and involuntary. However, Neumann (1984) and Carr (1992) pointed out that there are rare cases where all three features hold simultaneously in an automatic process. Others have introduced a minimalist (common feature) definition of automaticity. Logan and Cowan (1984) introduced the concept of ballisticity. A process is said to be ballistic if it completes its task, once started, without involvement of conscious control (an act of will). This identification of conscious control with an act of will confounds consciously mediated action with voluntary action. Bargh (1992, 1994) pointed out that ballisticity is common to all automatic mental process and proposed that it be adopted as the definition of automaticity. Logan (1992) argues that a common feature for automaticity is that it depends on attention and is a postattentive process. We agree with Logan's view. Each step of a novel task (before it is automatized) is processed with the involvement of consciousness. In global workspace theory, attention, which is initiated voluntarily or automatically, controls access to consciousness. At any given point in the processing of a task, attention brings the necessary information content (knowledge, skill, etc.) to consciousness. The consciousness processes use the information content to recruit the necessary computing resources to perform the next step of the task. It is our view, whether a task is automatized or not, that a task is initiated with the service of the same attention system (an attention codelet). In our mechanism, a task is operated in the same sequence using the same set of computing processes (codelets or low-level action units) whether it is performed automatically or under conscious control.

Tzelgov (1997) proposed that automaticity be defined as processing without monitoring in the sense that monitoring is the intentional setting of the goal processing and the intentional evaluation of its output. We do not have a disagreement with this definition, except that we qualify monitoring to be conscious monitoring or consciousness. Also, we believe that an automatized task can be initiated voluntarily or intentionally. This comes from our assertion that if a task is initiated as a result of a voluntary attention, then the same voluntary attention (intentional setting) is needed to initiate the task after automatization. That is, once automatized, the triggering intentional setting ("what to do") is always consciously experienced while the triggered goal structure ("how to do") that produce the actions is automatic.

In agreement with Logan's definition, automatization in our mechanism is dependent on attention and, as such, is a postattentive phenomenon. In our mechanism, different attention components, in competition with each other, try to bring the corresponding pieces of information content (knowledge, skill, etc.) to consciousness. The attention component that wins the competition brings its content to consciousness to be broadcast to other processes. Our belief is that automatization, as a postattentive process, reduces the competitive strength of the relevant attention components. As a result, the role of consciousness fades in the processing of the task. Automatized tasks are not affected by the serial and limited-access constraints associated with consciousness, and they can be performed in parallel.

Bargh (1997, 1999) stressed the importance of automatization, stating that much of psychological processing (perceptual, cognitive, social domains and implicit learning) is automatic. Automatization frees up one's limited conscious attentional capacity/resource from automated tasks in which it is no longer needed (Barge, 1999 and many other before him). Automatization is itself implicit learning, which takes place automatically without our conscious awareness. Although implicit learning is automatic, it requires conscious information (Reber, 1989). Conscious regulation of one's decision and action is effortful, requires limited resources that can be exhausted very quickly, and is a serial process that is relatively slow. In contrast, automatization allows an effortless, relatively fast, and mostly parallel control of one's own behavior and decisions. If we were conscious of everything that we do in our every day regular activities, we would not have the time and resources to perform tasks that are deemed important. By not bogging down consciousness with what we call trivial routine tasks, the freed consciousness is used to solve problems requiring informational input and for creativity. Without automatization, no task will be trivial.

One disadvantage of automaticity is that it is not easy to turn off automatized skills and automatized processes could produce action errors by being active in the wrong contexts. Also, arguably a disadvantage, we lose the details of our automatic skills. If needed, the steps in automatized tasks can be observed voluntarily (be aware of what we do automatically), but this defeats the main advantage of automaticity: to save the important resource of consciousness. So, automaticity is inherently inflexible.

5.3 Requirements for Automatization Mechanism

In an agent system, the execution of a behavior produces actions that bring about a change in the internal state of the agent or in the state of the environment, or both. A sequence of behaviors implements an action plan for a task whose execution transforms a given initial state into some goal state. In other words, a task is a particular process of achieving a desired goal in a given situation. A particular task could involve a different sequence of behaviors at different times. A repetitive task that gets carried out with the same sequence of behaviors becomes routine and with time. The skilled control of action (automatization) is realized by a high degree of synchronizations and coordination in the interaction of processing units.

In humans, automatization is a continuous process. We observe its effects all the time. For automatization to happen, a task, which gets applied to a particular situation, needs to be performed with consistency or with the same sequence of behaviors every time. Many human activities such as walking, swimming, cycling, driving, flying a plane, (in general all competitive sports), playing a musical instrument, etc. involve some type of automated action as we practice and acquire related skills. As we practice a task, we build/learn skills and as a result perform better; so goes the saying "practice makes perfect." That means automatization buys us improvement of performance. In the sports arena, given all other things being equal among the athletes, the winner is the one with the most practice.

Our work is concerned with the automatization of procedures that control actions, resulting from repetitive tasks that call for the execution of the same sequence of behaviors under the same situation. A task at the higher level of abstraction may involve actions that operate in different situations. For instance, the task "install a joint with a given bolt and nut" may use a set of actions that could handle the shape and size of the heads of the bolt and nut and the appropriate wrench to be used. The size and type of heads of the bolt and nut and the implied type of wrench are the variables of the behavior, which need to be bound for its execution. Although the high-level task is the same, the set of low-level actions under behaviors with variables varies depending on the bound values of those variables. A behavior with no variables (or with built-in variables) always uses the same set of low-level actions. *In this paper, we assume that behavioral automatization can happen when a task involves a sequence of behaviors, each with no variables or with all its variables that need to be bound by "consciously" mediated information are bound a priori.* Not having the need to bind variables with "consciously" broadcast information encodes the consistency of the environment under which a goal context executes.

What is the mechanism for automatization? How does automatization buy us performance? We will try to address these and other questions. First we have to know the conditions for automatization to happen. In general, the process of automatization starts with the recognition that a task is routine/repetitive, this happens during design time and/or via learning. In our agent architecture, automatization happens under the following assumptions.

1. The automated task is relevant in a given set of situational values or context.
2. The task is performed by the same sequence of behaviors all the time.
3. Each behavior in the sequence deals with information content, which is judged by the agent's awareness to be invariant or constant - has no variables to be bound from a consciously broadcast content.

5.4 Automatization Mechanism

5.4.1 General Behavioral Automatization from a Consciously Mediated Task

To execute a new action plan, one should have a conscious experience of the effects of its individual actions as the action-plan is being carried out. As the same action-plan executes repetitively (in some relatively short time interval), it is less likely that the effects of the actions get observed consciously. To illustrate the process with a parts-assembly example, a task that installs a car door in a car assembly line may have an action plan with instances of behaviors that have an execution order like "observe car position," "align door position," "get bolts," "put bolts in positions," "get power wrench," "drive bolts in position to be tight," and "put back wrench in place." A behavior stream performs such an action plan. The execution of a sequence of behaviors within a behavior stream may change from time to time. This is because a behavior stream can have two or more behaviors that are independent of each other and can be executed in any order. That is, a behavior stream can produce more than one sequence of actions in order to perform its intended task. For an agent that is new in the example task in the assembly line, there is a "conscious" effort on the part of the agent to determine the location of the relevant assembly parts and tools at a given time, and to consciously be aware of the effects of the behavior that was just executed. In performing this task, time and energy is expended in the "conscious" observation of the working environment and execution steps. If the agent in this example repeats this task with the same sequence of actions successfully, as time passes, the process of accomplishing the task happens with less involvement of consciousness. With enough practice or repetition of the same task, the agent executes a part of its behaviors unconsciously. That is, behavior-level automatization develops via learning.

As stated in previous sections, in our agent architecture, everything is done by codelets. The actions of high-level constructs are for the most part executed by codelets. Codelets jump to the playing field in order to affect the behavior of the agent. Codelets that are active together in the playing field build association between them (Jackson, 1987). Our task here is to build a mechanism that will realize the automatization process by enabling behavior codelets, by means of their associations, to recognize the presence of routine and repetitive sequences of behaviors while executing a particular task. When a task is performed, behavior codelets and relevant attention codelets reside in the playing field at the same time and build associations among them. When the task is performed repetitively with the same action plan schema, the associations among the different codelets relevant for the task get stronger. Also, there is a competition process among attention codelets. Attention codelets compete for access to consciousness on the basis of their activation. These *competition* and *association* processes are the basis of the automatization mechanism we present here.

Conscious awareness can be related to a behavioral action in different ways: an intention that initiates a behavior, an attention to the causes of a behavior, an attention that a behavior is under execution, and an attention to the effects of a behavior. Intention codelets try to bring to "consciousness" a behavior as a goal image and for possible conscious goal selection. Attention codelets can watch and try to bring to "consciousness" the causes of a behavior or the status of the behavior under execution. Expectation codelets can try to bring the outcome of the behavioral action to awareness. So there are multiple ways for a behavioral action to become a conscious experience. In what ever way a conscious experience does come about, behavior codelets and attention/intention/expectation codelets that are involved in a consciously mediated task will build associations while they are active in the playing field.

5.4.2 Conditions for Automatization

In our mechanism, whether a behavior/task is triggered voluntarily or automatically, the same underlying processes or codelets are used. The automatization process itself is automatic, and thus is a type of implicit learning or skill acquisition. In terms of our architecture, the conditions for automatization are as follows.

Behaviors or behavior streams need to be primed for instantiation in one of two ways: (1) via the broadcast mechanism, which is the only way before automatization and (2) via association links among behavior and attention codelets that develop with the automatization process. The association links allow exchange of task and control information among the codelets. Codelets build associations while they are active at the same time or within a relatively short time window. Considering any two successive behaviors in a sequence of executing behaviors, a behavior codelet under the leading behavior can build an association with the behavior codelet under the trailing behavior. If there is enough association between such two behavior codelets, the first behavior codelet primes the second behavior codelet. In other words, the association strength between two behavior codelets is the basis for the automatic priming. Figure 5-1 depicts the priming effect of the associations among codelets. Attention codelets also develop strong association links to behavior codelets by being active together. The order of the actions in a given task must remain the same with or without automatization. That is, the sequence of actions in an action sequence must be independent of the priming method. This point of view is supported by Bargh's (1997) assertion that goals/sub-goals operate in the same way regardless of how they were instigated (automatically or "consciously").

When behavior priming of type (2) happens due to automatization, the likelihood of behavior priming of type (1) should decrease ("consciousness" should be saved for other more important duties). This is to say that the competitive strength of the relevant attention codelet fades and, as a result, is less likely to gain access to "consciousness."

We assume that automatization happens only for actions (behaviors and the associated codelets) that operate in a consistent domain environment; or responses to situations do not involve dynamic variable binding using “consciously” broadcast contents.

Next, we will describe how the above conditions are met in the automatization mechanism.

5.4.3 Details of the Mechanism

When any two codelets are together in the playing field, the association between them becomes stronger (or weaker if things are not going well). At the beginning of the run of a hypothetical task, suppose an attention codelet (let’s call it AC1) jumps to the playing field bringing with it the information that will eventually cause the starting of the task. When the attention codelet makes it to “consciousness,” its information, with that of its coalition, is broadcast to all behavior codelets. A behavior codelet (BC1) that found the broadcast relevant binds itself with the broadcast information and jumps to the sideline from which it primes its behavior stream. A primed behavior stream, which is the action-plan or part of the action-plan, gets instantiated if it is not already instantiated. The behavior net mechanism eventually picks a particular behavior (B1) for execution, and its behavior codelets jump to the playing field. For the sake of discussion in this section, let’s assume that each behavior has only one behavior codelet. Then, B1 will have its codelet BC1 in the playing field executing its piece of the task. The action of BC1 may or may not attract the interest of any attention codelet. But, for this particular task, let’s assume that the action of BC1 causes attention codelet AC2 to jump to the playing field and eventually come to “consciousness.” Then the broadcast of its information content happens.

In a similar fashion, a behavior codelet (BC2 under behavior B2) finds itself relevant to the broadcast information (of AC2) and then jumps to the sideline where it instantiates a behavior stream, if necessary, and binds variables in behaviors. The behavior net mechanism eventually chooses behavior B2 for execution and its behavior codelet BC2 jumps to the playing field.

In the hypothetical task we considered above, suppose a sub-task is executed by the actions produced by behavior codelets BC1 and BC2. Suppose also that BC2 has no variables to be bound, thus satisfying one of the conditions for automatization that we discussed above. Observing the arrival of codelets in the playing field, we find a time sequence of BC1-AC2-BC2. Suppose our hypothetical task has to be executed repetitively, its sub-task is done “consciously” and produces the action-“consciousness”-action sequence of BC1-AC2-BC2 repetitively. As automatization builds, the associations BC1-AC2, BC2-AC2, and BC1-BC2 increase. When some of these associations have strength over threshold, automatization can take place. For our discussion in this section, let the association strength from A to B is given by *association(A,B)* and let $D \in \Re$.

5.4.3.1 Predicting the Next Behavior in a Sequence

As a result of their association being over threshold, BC1 is able to spread activation energy to BC2 directly and as a result BC2 primes its behavior B2 with the probability of the activation energy received; this means priming happens without broadcasting the content of AC2. BC1 sends activation energy directly to BC2 in proportion to $\boldsymbol{x}$ = *association (BC1, BC2).*

The actual priming activation energy A_p from one behavior codelet to another one is computed using equation (5.1),

$$A_p = A/(1+e^{-ax+c}) \tag{5.1}$$

where $A \in R^+$ is maximum activation energy (usually set to 1.0), $a \in R^+$ denotes the rate of increase of priming activation within the interesting range of *x,* and $c \in R^+$ denotes the threshold value for association *x* and shifts the priming activation function on the x-axis to the right or to the left.

In equation (5.1), we can see that the priming activation increases towards the maximum possible as the association *x* increases over the threshold c; it decreases as *x* decreases towards zero. The ability to predict the next behavior in a sequence is the result of the learning process in the automatization mechanism. In this case, the learning takes place by changing association weights (the weight-based representation) between codelets.

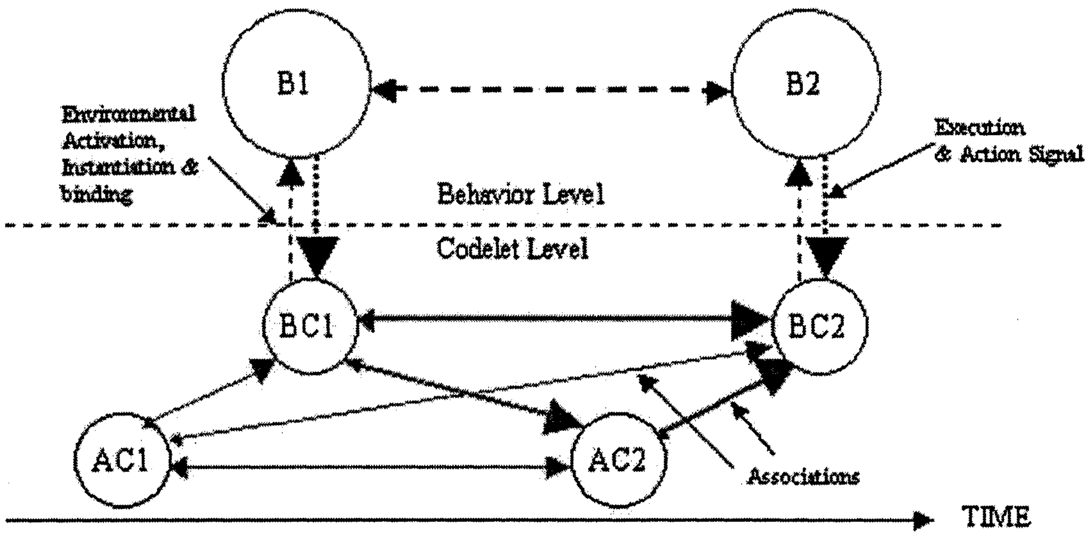
B1
B2
Environmental Activation, Instantiation & binding
Execution & Action Signal
Behavior Level
Codelet Level
BC1
BC2
AC1
AC2
Associations
TIME

Figure 5-1: Automatization mechanism: AC1, AC2 – attention codelets, BC1, BC2 – behavior codelets, and B1, B2 – behaviors.

As Jackson (1987) suggested, the weight of associations between codelets in the playing field adjusts asymmetrically based on the time of arrival of codelets in the playing field. Link weights from codelets that arrive before time T to codelets that arrive after time T are more strongly adjusted than those from codelets that arrive after time T to codelets that arrive before time T. This temporal effect on association strengths helps to encode coordinated sequence of actions. Once strong links are established between codelets, a coordinated sequence of action could be initiated when the first set of codelets in the sequence join playing field. The set of codelets then excites, via the strong links in the sequence, the next set, which, in turn, excites the next set in the sequence, and so on. The learning (strengthening of links) associated with a task sequence can develop from performing the task repetitively. With repetition or practice, the system tends to coordinate/synchronize the sequence in the task automatically. In what ever way the priming happens, the associations allow one codelet in the playing filed to activate another strongly associated codelet to join the playing field, and continuing the sequence in the same fashion is what Jackson (1987) calls *associative engine*. Predicting the next behavior in a sequence is underlain by the associative engine mechanism. This fact is supported by the study of Chartrand and Bargh (1996), which shows that primed information-processing goals operated the same way as did consciously and intentionally activated goals.

The passing of priming activation is based on strong associative links in a temporal sequence and is not limited to happen between behavior codelets. Referring to Figure 5-1, AC2 has a strong association with BC2 and therefore AC2 can prime BC2. The important point is that a behavioral action processor or a processor associated with the effect of the behavioral action primes a processor of the next strongly associated behavioral action.

5.4.3.2 Lowering the Intensity of Attention

Referring to figure 5-1, the attention codelet AC2's effective activation is diminished as automatization builds so that it has less probability to make it to "consciousness." Here, we compute the effective activation of AC2 by multiplying the maximum activation level (*Ar*) with that of *x,* which is the product of the strength of associations: *association (BC1, AC2)* and the *association (AC2, BC2).*

The effective activation level (A_{eff}) of an attention codelet AC2 is computed using equation (2).

$$A_{eff} = A_r\left(1 - \frac{1}{1 + e^{-bx+c}}\right) \quad (5.2)$$

Where A_r is the maximum activation level of the attention codelet AC2 (that is, the activation level of AC2 without the effect of automatization), $b \in R^+$ denotes the rate of decrease of the effective activation level of an attention codelet within the interesting range of *x*, and $C \in R^+$ is a threshold that shifts the activation level function on the x-axis to the right or to the left.

In equation (5.2), when there is no matured automatization (*x* is near zero), the attention codelet AC2 can attain its maximum activation level that is near to A_r so that it could compete for "consciousness" at its full strength. As automatization gets strong or *x* increases, the effective activation of the attention codelet AC2 approaches zero so that it will not come to "consciousness." Damping down the activation strength of attention codelets is another learning process in the automatization mechanism. Here, we can say that learning changes the activation-based representation at the behavior codelets.

5.4.3.3 Action with and without "Conscious" Involvement

With the automatization process, as a result of associations among codelets, an agent system can act with and without "conscious" involvement.

As discussed above, the priming effect on a behavior happens via an associative activation link from one codelet to another. After automatization, there will not be a broadcast instigating this next action (the lowered activation level of AC will make it unlikely that it will come to "consciousness"). In Figure 5-1, the action of behavior B1 would not be "consciously" observed if the automatization process effectively lowered the activation level of attention codelet AC2 so that AC2 cannot win the competition to make its way to "consciousness."

On the other hand, even with a well-matured automatization, an action could be observed "consciously." If there is nothing interesting going on, there is a possibility that an attention codelet whose activation level is damped down due to automatization could still make it to "consciousness." That is, a behavior codelet gets its relevance via "consciousness" (broadcast) or via direct association of codelets (automatization) or via both (when "consciousness" is relatively idle). This type of situation could happen in humans; one can "consciously" observe his/her already automated actions under a situation where there is nothing better to do.

5.4.4 Experiencing Instances

In our mechanism, one of the conditions for automatization to occur is that the behaviors and codelets involved in the automatized task should not have variables to be bound with contents of "conscious" broadcasts. Does this mean that tasks that involve variable binding can not be automatized? For an infant, do not all the behaviors, except the innate ones, have variables to be bound? Is this condition far fetched? We will try to show the validity of this condition by succinctly answering these questions. This does not mean that tasks with variables can not be automated. But, in an acceptable level of granularity, if a variable binding is necessary to perform a task, that task can only be done with the involvement of consciousness. Once a variable is bound, the active codelets and behaviors that perform the task are instances that do not have any "consciously" observed changeable values. Instances, with all variables bound, are constant entities. Automatization happens only with such instances of codelets. Looking at this differently, the constant aspect of instances is one dimension of the consistency, which is an important condition for automatization to happen, the other consistency being the order of actions and interactions among the relevant instances of codelets and behaviors during the performance of a task.

The general task of driving can illustrate this point. One can not practice with general driving task schema. Practice can happen only with the instances of processes (codelets) associated with a particular car one drives. The automatization of driving happens with the instances that are already bound with particular values (the level of pressures during acceleration and braking, number of gears, level of resistance on the steering wheel, etc.) of a car. If a relevant parameter value of the car changes, it causes one or more new instances of codelets to become active. This introduces novelty and certain steps of the driving must be done consciously. Only after enough repetition or practice takes place in the changed situation and new automatization develops at the affected steps does the driving process becomes fully automatic again. There are a number of common skills that are used to drive different cars. If one builds all the skills needed to drive one type of car, he/she can easily automatize the task of driving a different car. This is because of the reuse of the many skills already acquired, or there are many common instances of codelets and behaviors that are used in the driving of the different cars. The difference between any two cars has a degree of granularity. Differences below some level of granularity will not be "consciously" noticeable and such differences are neglected. If the difference between two cars is negligible, one will not be aware of the difference and a single skill of driving, with the same instances of codelets and behaviors, can operate the two cars.

In conclusion, one can practice with instances of computing components (codelets and behaviors). Instances are created out of novel tasks when they are triggered and performed consciously. Then automatic processing develops when the instances of computing components are used in some constant order with enough frequency in a relatively short time span. So, it is not far fetched to assert the condition that automatization can not develop when dynamic variable binding happens in codelets and behaviors.

5.4.5 Forgetting Skills

We have seen how the automatization mechanism builds automatic skills. In general, if skills are not used for a relatively long time, they can be lost, either fully or partially forgotten. Computing resources in both biological and artificial worlds are finite, and forgetting is a mechanism of reclaiming unused or relatively rarely used resources. How can we accommodate this mechanism here?

The basis of the automatization mechanism, an implicit learning, is the associations that develop among instances of codelets in proportion to those codelets staying active together. In our mechanism, associations decay all the time, and the simplest way to forget skills is to wait until the associations decay sufficiently. But this is not a conservative method that will insure forgetting will take place. Also, if an instance of a codelet dies or gets uninstantiated, then the associations built around it and the resulting automatization are lost. But, how do we decide when to uninstantiate an instance of a codelet? Our approach is to introduce a decay mechanism for each instance of a codelet with a variable decay rate. The decay rate depends on the frequency and recency with which the instance of the codelet was active. Higher frequency and higher recency values means a lower decay rate and vice versa. An instance of a codelet with higher decay rate dies faster and an instance of codelet with lower decay rate stays instantiated longer. Frequency is directly proportional to the strength of association the codelet has with other codelets, and recency is inversely proportional to the length of time since last it was active. Each of the frequency and recency parameters has a coefficient to tune its relative importance in the computation of the variable decay rate.

One could argue that there are cases when skills like swimming will never be lost once they are internalized. Although our mechanism can accommodate this situation, for practical or computational reasons, we do not strive to make it a standard property in our agent system.

5.5 Deautomatization Mechanism

5.5.1 Conditions for Deautomatization

We need automatization for efficiency reasons, primarily to save an expensive resource, "consciousness," but also to shorten the average time needed to execute a routine task.

The need for undoing automatization or deautomatization occurs when an automated action sequence or subsequence fails to perform as expected. The origin of the failure could come from many sources: change of environment, change of domain, a problem with a codelet or any combination of these. When a failure happens, the task is no longer routine and the automated sequence of behaviors that performs the task is no longer efficient. In this situation "consciousness" is again needed, and our mechanism should provide a way of suspending automatization. In this process, the first problem is to detect the failure of the automated task.

5.5.1.1 Detecting Failure

A failure of an automated task happens when one of the behaviors in the sequence does not produce its expected outcome. The expected outcome of a behavior is decided by the propositions in it's add and delete lists. To detect a failure, there needs to be feedback on the effect of the behavior's action. Expectation codelets of an active behavior attempt to provide this feedback by bringing to "consciousness" the failed result of the behavior codelet(s) that work under the behavior. To detect failure is to observe a mismatch in the comparison of the actual outcome of the action as observed by the expectation codelet(s) with the expected outcome in the specification of the behavior. Failure is detected if a mismatch exists between expect outcome and actual outcome of a given proposition. Detection of failure by expectation codelets may require multiple cognitive cycles.

5.5.1.2 Failure Propagation

In executing an automated task, a failure could occur at any action in the sequence of behaviors. Any failure calls for the involvement of "consciousness." Particularly, there is a need to involve "consciousness" in the execution of all actions in the sequence that precedes the point of failure. Referring to figure 5-2, a task/subtask executes a sequence of three behaviors B1, B2, and B3. Suppose a failure is detected at behavior B2. To remedy the failure, both the actions of B1 and B2 need to be monitored "consciously." To reinstate "conscious" control, the effect of automatization on the attention codelets AC1 and AC2 has to be removed temporarily. For this to happen, the mechanism should propagate the failure status from the behavior B2 to the preceding behavior B1.

As part of the mechanism and in relation to failure handling, expectation codelets detect the failure of a behavior codelet's action and signal the failure status to its sibling behavior codelets by bringing the failure to "consciousness." On receiving a failure signal, a behavior codelet passes that signal to all predecessor behavior codelets with which it has strong associations. With this mechanism, a failure status propagates from the point of failure to the preceding behaviors via the strong association of behavior codelets that happens during automatization. Next, we will see the details of this mechanism.

5.5.2 Mechanism to Suspend Automatization

To remove the effects of automatization, our mechanism should somehow disable the priming of a behavior via the association of behavior codelets and should disable the suppression of the effective activation level of involved attention codelets. We will describe how this happens.

5.5.2.1 Undoing the Effect of Priming Activation Energy

Each behavior codelet (BC) has a suspension parameter $S \in [0,1]$ with an initial value of zero. As a process of the mechanism, each behavior codelet spreads activation energy to every other associated behavior codelet according to equation (5.1). This suspension parameter value changes the sensitivity of BC to the level of the priming activation energy it receives.

Equation (5.1) is modified to equation (5.3) that shows how the suspension parameter S changes the effective priming activation energy (A_p) that spreads from one BC to another BC.

$$A_p = (1 - S)A \big/ (1 + e^{-ax+c}) \qquad (5.3)$$

$S = 1$ means that priming by association is suspended at BC, and $S = 0$ means that priming by association is not damped down in any way. The S value between 0 and 1 gives the level of sensitivity of BC to received priming activations from associations.

When a failure happens after the execution of a behavior (its behavior codelets), each behavior codelet under the behavior sets its suspension parameter value to one; in effect a suspended behavior codelet disregards a priming activation that comes from all associated behavior codelets. In terms of the pandemonium theory (Jackson 1987), a behavior codelet in the stands stops hearing the yelling of other associated behavior codelets in the playing field.

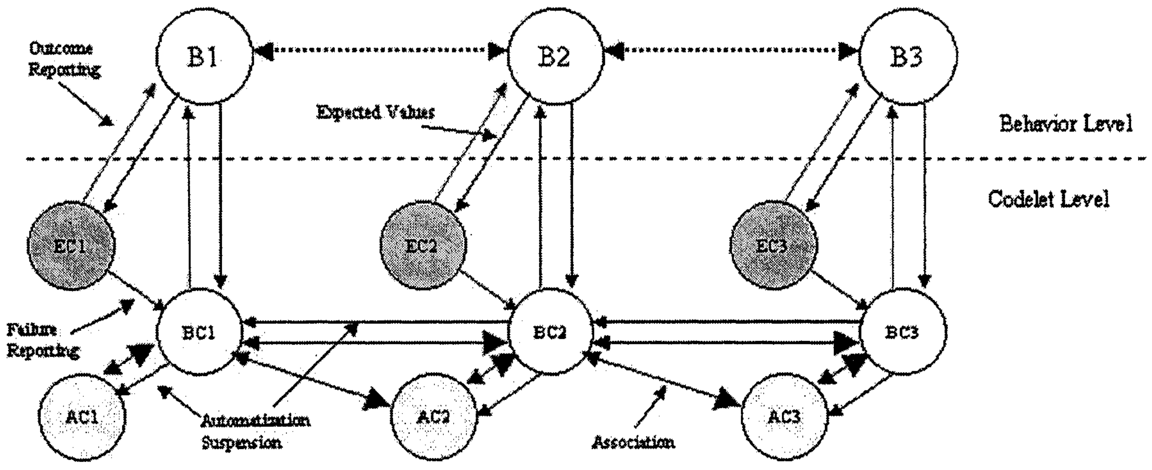

Figure 5-2: Deautomatization mechanism (also shows the automatization mechanism); expectation codelets EC1, EC2 & EC3 respectively underlying and controlling behaviors B1, B2 & B3.

5.5.2.2 Undoing Suppression of Activation-Level of an Attention Codelet

Each attention codelet also has a suspension parameter $S \in [0,1]$ which has an initial value of zero. Also, each active behavior node in the behavior net system has its own expectation codelet (EC), or possibly more than one. When a behavior becomes active and performs its action, the expectation for the outcome of the action is set by activating its ECs. The ECs monitor the effects of the execution of the sibling behavior codelets (BCs) and provide feedback on the failure or success of the behavioral actions. If the expected outcome of actions matches with the actual outcome of actions, then the feedback is a success. Otherwise, the feedback is a failure, which could be a partial or a full. The ECs try to bring the feedback information, whether the expected outcome is fulfilled or not, to "consciousness" by jumping to the playing filed with an appropriate activation level. As part of the self-organization mechanism of the system, the EC acts to disable the effect of the automatization in the situation where the expected outcome is not fully attained by the behavioral actions. The action of EC is simply to signal a sibling BC so that each such BC disables the effect of automatization on the associated attention codelet. The BC disables the effect of automatization by somehow removing the effect of the association between itself and the attention codelet AC and consequently the amount of activation that spreads from BC to AC weighted by the association strength. Particularly, in our implementation, BC, when signaled by its sibling EC, sets the suspension parameter of each of the associated attention codelet to"1."

Upon receiving the signal to disable the effect of automatization, a behavior codelet (BC), not only passes a signal to its associated attention codelet (AC) to set its suspension parameter, it also passes a signal to its predecessor behavior codelets (with which it has strong association as a result of automatization) to disable the effect of automatization at its end if there is any. Therefore, when a failure of an automated action sequence is discovered by an expectation codelet of the behavior at the failing point, the mechanism depicted in figure 5-2 uses the predecessor links to set the suspension parameters of all attention codelets that are precedent to the expectation codelet to "1."

In this mechanism, ECs try to bring the feedback on the outcome of the behavioral action to "consciousness." This is a conscious interaction to share the outcome of behavioral actions globally. The mechanism also uses automatic or unconscious interactions among the different codelets (ECs with sibling BCs, BCs with strongly associated ACs, and BCs with non-sibling and strongly associated BCs) to signal and propagate failure status of behavioral actions as part of the deautomatization process. The unconscious interactions are supported by the theory we are implementing. Global workspace theory states that most of the actions and interactions of processors takes place unconsciously. Unlike conscious broadcasts – a serial and limited-capacity interaction channel, unconscious interactions are parallel and over a wideband communication channel.

From the above discussion, we can see that the suspension parameter is introduced to suspend the effect of automatization when a failure is detected in an action sequence. To do so, we have to come up with a formula to compute the effective activation level of an AC so that a task can be carried out via the "consciousness" process when an automated action sequence fails, thus the undoing/neutralization of the automatization.

Equation (5.2), which computes the effective activation (A_{eff}) of an AC, is modified as shown in equation (5.4) so that it can incorporate the undoing of the automatization.

$$A_{eff} = (1 - S)A_r\left(1 - \frac{1}{1 + e^{-bx+c}}\right) \qquad (5.4)$$

An expectation codelet (section 4.3.3) is a type of attention codelet, and as such, an expectation codelet behaves like an attention codelet in the sense that: it tries to bring information to "consciousness", it builds association with other codelets, and it participates in an unconscious low-level intra-codelet communication. The only difference is that an expectation codelet always sets its effective activation level to the maximum whenever it detects an error. That is, an expectation codelet overrides the effects of automatization in an error situation. This allows the mechanism to reinstate conscious control when an error is encountered during the execution of automatized skills.

5.5.2.3 Forgetting Suspension of Automatization

Once automatization is suspended, all actions of the task are executed with the involvement of "consciousness." After a while it may be desirable to reinstitute the automatization of the task since the source of the failure may not exist any longer. The automatization process cannot take control unless the suspension parameters are reset. That is, the system needs to forget the setting of suspension parameter. Decay is a basic built-in mechanism that forgets changes in parametric values. As it is the case for the activation level of codelets and association strength between any two codelets, the "suspension" parameter values at the different codelets decay with time. The deautomatization effect has a relatively short time span, and so the decay rate is set high. The characteristics of the decay can be tuned by controlling the parameters of the decay curve.

5.6 Effects of Practice

Performance improves with automatization or practice. That is, speed of processing a task increases with practice. Numerous studies have demonstrated that improvement of performance follow a *power law of practice* and is common in many human tasks: perceptual-motor skills (Snoddy, 1926; Crossman, 1959), perception (Kolers, 1975), attention and perceptual learning (Shiffrin & Schneider, 1977), motor behavior (Card al. et., 1978), memory (Anderson, 1980), routine cognitive skill (Moran, 1980), and problem solving (Neves & Anderson, 1981; Newell & Rosenbloom, 1981). This law, also known as the *log-log linear learning law*, states that the time to perform a task decreases as a power-law function of the number of times the task has been performed. Our approach here is to show the qualitative improvement of performance with the aspect of intelligent behavior associated with practice; we do not claim to model the empirical behavior of the power-law.

We can discuss how performance improves in our mechanism by singling out the used or unused steps of the cognitive cycle of IDA in performing automatized procedures (see Figure 5-3). In performing an automatized procedure, suppose we start at an action of a behavior B1 whose behavior codelets and expectation codelets are executing their task (cognitive-cycle step 9 - CC-9 with B1). At this step, the behavior codelets complete their execution, and then the expectation codelets monitor the outcome of the action. Suppose the action caused a change in the environment, the monitoring by the expectation codelet will not be completed until the environmental change is reflected in the working memory. This needs the service of steps 1-3 of possibly a subsequent cognitive cycle. The change in the environment is perceived, the perceived information is stored in the preconscious buffer, and local associations are made using the content of the preconscious buffer as a cue. Suppose the action is successful, the expectation codelets of B1 will observe the success information that is reflected in the working memory. That means, after CC-3, the expectation codelets of B1 can provide feedback on the action of the sibling behavior codelets. At this point, because of the strong association, the codelets that underlay B1 will directly prime the coalition of codelets of the next behavior B2 – this step might be considered as the step of recruitment of resources (CC-6). Once primed, the codelets that underlay B2 start their action by jumping into the playing field – CC-9 with B2. Similarly, the process continues in performing the sequence of behavioral actions of a procedure.

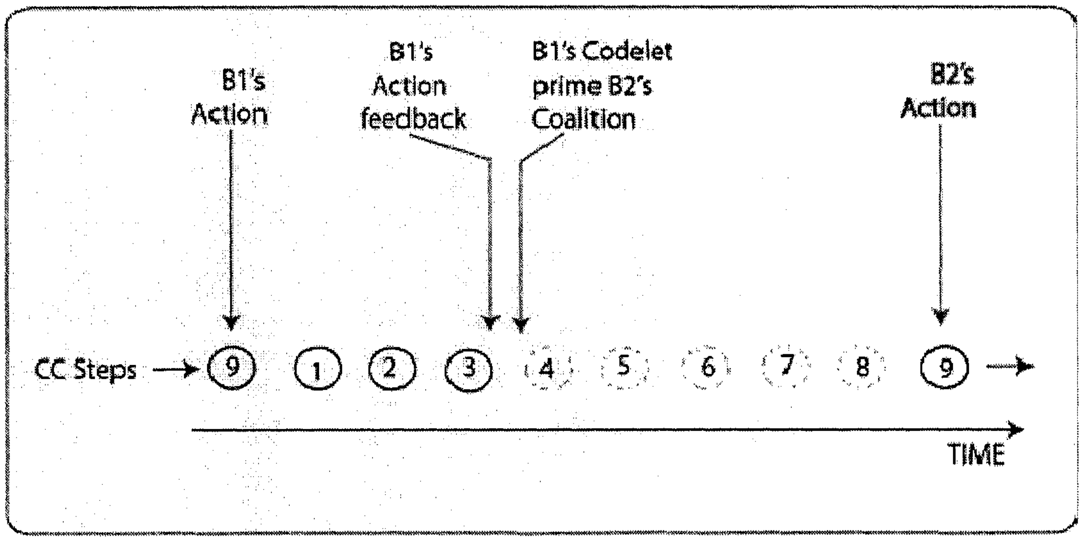

Figure 5-3: Cognitive cycle steps 1, 2, 3, and 9 are involved in performing automatized task. Between steps 3 and 4, feedback on the behavioral action of B1 is ready and then a codelets underling B1 can prime coalition of codelets underlying behavior B2.

From this analysis, cognitive cycle steps 4, 5, 6, 7, and 8 are avoided in performing automatized tasks. We can argue that the broadcast of “conscious” content (CC-5) is a serial process for a global dissemination of information and CC-5 would be a relatively time-expensive step. We also argue that expensive time complexity arises in cognitive steps that involve competition - competition for consciousness (CC-4) and choosing of a behavior in the behavior net dynamics (CC-8). Thus the automatization mechanism improves performance in executing automatized tasks by avoiding certain steps in the IDA cognitive cycle and, in that, the ones with high time complexity.

The automatization mechanism, as a learning system for tasks, hypothesizes that knowledge of tasks is internalized by associating intention, action and expectation (or attention to consequences) processors, which are involved in performing the task. This is an implicit learning using an efferent binding process (Engel et al., 1999).

5.7 Implementation Details

From our discussion in the previous sections of this chapter, one can see that automatization/deautomatization requires additional machinery. We have also mentioned that automatization/deautomatization happens automatically and the associated mechanism is part of the fixed system architecture. The automatization process changes the weight of links between low-level processors – codelets. Plus, different non-linear mathematical parameters are introduced to control the automatization/deautomatization mechanism. Also, different codelet types are required to have additional capabilities. For instance, behavior codelets are required to pass priming activation to others in proportion to their strength of association and also required to propagate automatization suspension signals to other strongly associated codelets. Expectation codelets are required to generate automatization suspension signals when an expected action outcome is produced, and attention codelets are required to compute their effective activation level based on their association strengths with behavior codelets. An expectation codelet is a special case of an attention codelet in the sense that it tries to bring action outcome, whether success or failure, to "consciousness." All codelets also must register their time of joining the playing field, so that building of asymmetric associations is possible.

5.7.1 Automatization Starter

Automatization starter (AutomatizationStartup.class) is an addition to the behavior network system in order to support the automatization mechanism. Its primary role is to create and build associations among co-active codelets and to provide necessary functions that are needed to characterize the associations. In the current implementation, association links grow and decay in a non-linear fashion and we will say more about them in a while.

The program is begun by the *behavior network starter* of the action selection system discussed in chapter 4. After initializing different parameters that it uses, this daemon program runs the following loop.

1. Build associations among codelets (with relevance to automatization) that are currently in the playing field.

2. Purge dead-ender associations. Associations become dead-ender if any one of the associated codelets has decayed and died.

3. Decay associations with time.

5.7.1.1 Building Associations

To build associations, the Automatization Starter first samples the playing filed for the list of active codelets. The association among the observed active codelets is built nonlinearly and based on the playing field joining time of the associating codelets. We use the general logistic function (lf) shown in equation 5.5 to control the growth of associations. In the equation, we can see that we can control three linear parameters: (1) amplitude of function (*A*), (2) coefficient to function variable (*a*), and (3) horizontal shift setting to function (*c*). An interesting design parameter is the x-value that maps to middle point of the function (X_{MP}), which is the transition point to strong associations. We decide the characteristics of the function by determining three parameters: Amplitude (A), coefficient (a), and middle point (X_{MP}). The parameter "c" is computed with the equation $c = a \times X_{MP}$.

$$lf(x) = A/(1 + e^{-ax+c}) \qquad (5.5)$$

Figure 5-4 shows a logistic function curve with parameters set to $A = 10$, $a = 0.8$ and $X_{MP} = 5$. The parameter "c" is computed: $c = 0.8 \times 5 = 4$. In our application of the curve here, the domain and range values are association weights. The coefficient (b) determines the growth rate of the associations. The amplitude (A) determines the maximum value of association weights. The middle point x-value (X_{MP}) determines the point where associations transit from weak to strong. At every learning step, the association weight is updated using equation 5.6. Δx is a small decimal value which is decided at design time. At each updating step, Δx is added to the previous value of an association, x, and the sum is processed by the logistic function (lf) to give the new association. We can set Δx to be nonlinear that changes as a function of the current association strength. The function is bounded by monotonically increasing function. So the association is always increasing, and it is bounded by the amplitude of the function.

$$ass_t = lf(ass_{t-1} + \Delta x) \qquad (5.6)$$

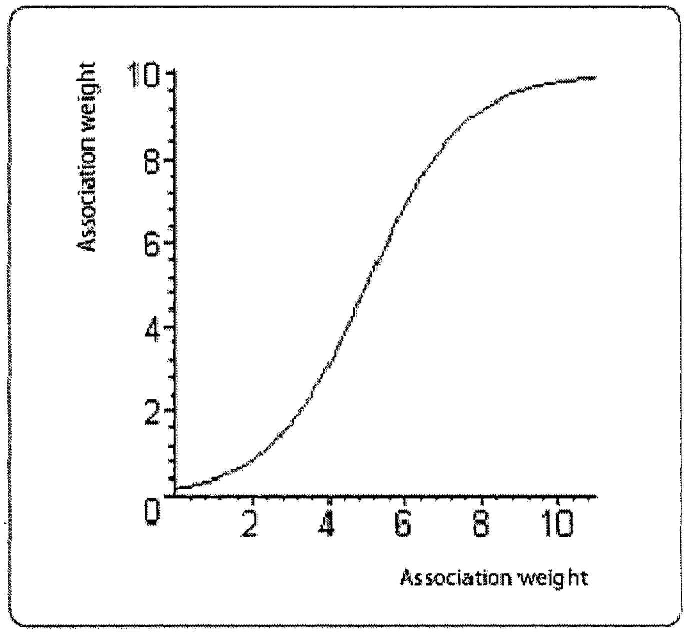

Figure 5-4: An example curve that governs the growth of association among coactive codelets.

5.7.1.2 Purging Dead-ender Associations

Associations are links between two instances of codelets, and instances of codelets get uninstantiated from time to time. For instance, behavior codelets instances decay and die if they have not been active for a long time. The dead codelet could be the head or tail of an association links. Once an association link has a dead-ender, whether head or tail, then the association link is not in use and the *Automatization Starter* purges it. In biological terms, if a neuron is dead, its synapses die with it.

5.7.1.3 Decaying Associations

The strength of the association encodes the degree of experience in the automatization mechanism – an implicit learning process. Strong associations imply strong experiences. A task skill developed via experience will degrade with time unless the skill is used regularly and recently. That is, forgetting increase with break in using skills or performing task. More over, well experienced tasks do not get forgotten or do get forgotten very slowly, and to the contrary, tasks that are less experienced get forgotten faster.

Forgetting has been a research issue in psychology for over a century (Ebbinghaus, 1885) and it is governed by Jost's (1897) law of forgetting, which states that if two memories are of the same strength but different ages, the older will decay more slowly than the younger. Despite, as reviewed by Wixted (2004a), there is no consensus on the underlying mechanisms or theories (decay: Woodworth, 1938; interference: Underwood, 1957, Underwood & Postman, 1960; consolidation: Plihal & Born, 1997; retrieval-failure: Tulving & Madigan, 1970) that are responsible for forgetting effects. Each theory provides its own possible explanation how Jost's forgetting law follows from the possible memory laying or learning processes (Wixted, 2004b).

Loftus (1958) advanced the idea that memory traces decay exponentially and a higher degree of learning results in a lower rate of forgetting. Also, Simon (1966) observed the possibility of higher degree of learning (a) induces a slower rate of forgetting and (b) underlies the Jost's law (1897) of forgetting - the older forgetting function yields a slower rate of forgetting. The empirical results from experimental psychology (Ebbinghaus, 1885; Underwood & Postman, 1960; others) are gathered based on initial explicit learning of associations followed by test of retentions after different intervals of break time. In automatization, an implicit learning, associations build as a task is practiced. That is, unlike the psychological experimental setups, here the implicit learning is online and degree of learning grows stronger with each practice run. The decaying (of associations or memory traces) curve, underlying the forgetting curve, is a continuous process and should integrate and interact with the associated learning curve. In the automatization type of implicit learning, we hypothesis that *degree of learning is encoded as the maximum association strength ever attained* between any low-level processors (codelets) that constitute a memory trace.

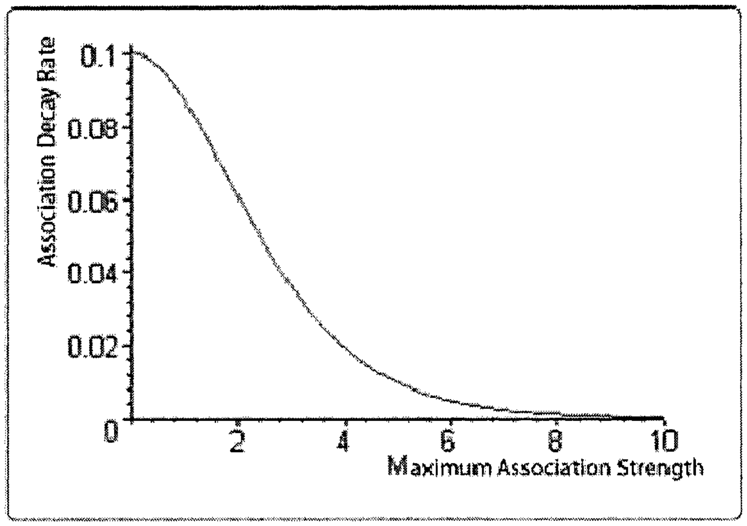

Figure 5-5: An example curve to govern decay rate for attained maximum association strength (degree of learning) between codelets.

As shown in figure 5-5, decay rate of association (of a memory trace) is a function of maximum attained association strength. The decay rates of associations underlie the decay rates of forgetting. As the degree of learning (maximum attained association strength) increases forgetting rate decreases (Loftus, 1958; Simon, 1966). There is continuous interaction of decaying/forgetting and learning curves means that the parameters that control the rate of decay and rate of learning need to be tuned so as to model empirical data (exponential or power curves) of practice and forgetting.

5.7.2 Direct Communication between Codelets

Global Workspace theory postulates that unconscious processes, including coalitions of processes that underlie goal contexts, can operate in parallel and can interact among themselves directly via a relatively wide bandwidth of communication channels. The automatization/deautomatization mechanisms, discussed in the previous sections of this chapter, entail interactions among behavior, expectation and attention codelets. We will briefly discuss the implementation of the interactions at these codelets.

5.7.2.1 Behavior Codelet Interactions

Behavior codelet instances interact with other codelets by sending and receiving event objects. Behavior codelets directly send/receive two types of events to/from other instances of codelets: (a) priming and (b) suspension.

Sending Priming Events

A behavior codelet, which is active in the playing field, can prime strongly associated behavior codelets via automatization (unconsciously), provided that priming activation levels (equation 5-3) have strength over a given threshold. Priming information is passed as event objects. A priming event object, as part of the automatization mechanism, carries pieces of information including a pointer to event sender codelet and the activation strength of priming. Priming activation energy is computed by using the logistic function given by equation 5.3. In the design phase, the parameters of this function should be tuned in relation to the design decision that determines the growth of associations among co-active codelets as discussed in section 5.6.1.

A behavior codelet sends a priming event to another behavior codelet if the computed activation energy is above a threshold. Although checking whether the priming activation energy is over a threshold is done at the sender end, it can well be done at the receiver end. This method is to have computational efficiency – few events are generated when the decision is made at the source.

At this time let's define a new relationship of codelets. Codelets that underlie a behavior and become active at the same time are called *sibling codelets or siblings*. That is, they join the playing field at the same time and leave the playing field almost at the same time, which means siblings build equal strength of associations to other non-sibling codelets. This also means that priming events generated by sibling codelets have an equal effect on any behavior codelet that they prime. As a result, a behavioral action primes another behavior via multiple priming pathways, and such redundancy is an expected feature of distributed computing systems.

Receiving Priming Events

For a behavior codelet, receiving a priming event signifies that a highly associated event sender codelet has been active in the playing field. Although, not used at the receiving end, the activation level signifies the strength of priming or strength of the automatic intentionality. In terms of the sequence of behaviors that perform an automatized task, the priming event is a signal for a receiving behavior codelet about the completion of action of a predecessor behavior in the task. That is, priming events, related to a predecessor behavior, are generated by codelets that underlie the predecessor behavior.

If a behavior codelet has received a priming event from a sender that underlies a predecessor behavior, then its siblings also receive a priming event from the same sender. This is again due to the fact that sibling codelets are always co-active and build the same relationships with other codelets.

Sibling behavior codelets, upon receiving priming events and validating that their preconditions hold, become active and execute the action of the behavior they underlie, which is the next behavior in a sequence of behaviors that perform an automatized task.

Along with the priming information, a behavior codelet may also receive updating information content from the priming codelet. This mechanism allows the binding of variables of behavior codelets with the fine details of situations that the fidelity of awareness may consider constant. The mechanism suggests that action processors receive information directly from information input processors and outside conscious control.

Propagating Suspension

The implementation uses a suspension event object to propagate suspension information. Every time a behavior codelet receives a suspension event, it does three things: (1) sets its suspension parameter to a value of 1.0, (2) propagates a suspension event to strongly associated predecessor attention codelets, and (3) propagates a suspension event to other strongly associated, predecessor behavior codelets. Only behavior codelets propagate suspension events.

5.7.2.2 Expectation Codelet Interactions

An expectation codelet directly interacts with other codelets by sending information about failure or success of actions.

Setting Suspension

We have already discussed that suspension to automatization happens when an expectation codelet of an active behavior detects unexpected outcome of action. If a failure is detected in an execution cycle, an expectation codelet does two things in relation to suspension of automatization: (1) sets suspension parameter to a value of 1.0, and (2) communicates the failure status to its sibling behavior codelets by generating and passing suspension events. Only expectation codelets generate suspension events.

Sending Priming Events

An expectation codelet is co-active with its sibling behavior codelets and builds similar associations with other codelets as its siblings do. And an expectation codelet can prime other behavior codelets that underlie successor behaviors in a sequence the same way behavior codelets prime, which we have discussed (section 5.6.2.1.1).

Expectation codelet passes a priming event only when an observed action produces the expected outcome and, of course, if there is a priming activation over a threshold. Expectation codelets, besides observing success or failure of an action, know the actual binding values associated with an outcome of action. This enables them to have more complete information about a completed action, and they can communicate enhanced priming information to predecessor behavior codelets.

If an expectation codelet does not observe a failure and none of its computations of priming activations are above the threshold, then it tries to bring the outcome to "consciousness."

As we have mentioned before, every time an expectation codelet detects an error, it always tries to bring the error situation to consciousness. That is, automatization has an effect on reporting success (via priming or via consciousness), but has no effect on reporting failure (only via "consciousness").

5.7.2.3 Attention Codelet Interactions

Attention codelets receive suspension events from strongly associated behavior codelets when failure of action is observed. They also pass information on the fine details and updates of its content to strongly associated behavior codelets.

Receiving Suspension

As we discussed earlier, behavior codelets, as they receive suspension events, propagate the suspension events to strongly associated, predecessor attention and behavior codelets. When an attention codelet receives a suspension event, it sets suspension parameter value to 1.0. The role of the suspension value is to modulate the strength of the effective activation level of an attention codelet. This role is specified in the inverse logistic function given in equation 5.4. An agent builder is required to determine the parameters of the inverse logistic function in relation to the domain. The domain is the product of two associations an attention codelet has. One is the strongest association from a predecessor behavior codelet to the attention codelet, and the other is the strongest association from the attention codelet to a successor behavior codelet. As associations get stronger with experience, the effective activation gets weaker and as a result, the less likely the attention codelet get access to consciousness.

The selection of the parameter values for this curve should be tuned in relation to other parameters used in the automatization/deautomatization mechanisms. For instance, one characteristic in the mechanism could be to have transition to matured automatic priming before attention codelets lose energy for their competition for conscious access. That is, to have automatization space in which automatic and conscious control could occur simultaneously.

Sending Priming Events

Attention codelets can prime behavior codelets with which they have strong association. An attention codelet passes priming information only if it is active and its priming activation energy towards a behavior codelet is over a threshold. Along with a priming event, an attention codelet could pass detailed information on situations it monitors – possibly at the level of detail that is not accessible for "consciousness." The low-level detail corresponds to direct sensory information that affects motor behavior. We earlier pointed out that automatization could happen when there is *invariance in the execution of a task*, and once automatized, behaviors do not need to rebind their variables. Do we contradict ourselves when unconsciously passed information is used to rebind variables of behavior codelets? We do not believe so. Let's discuss with an example of returning a ball in a tennis game. Suppose a player has an automatized skill to return an overhead ball. An attention (intention) codelet could prime and initiate a sequence behavior codelets that execute the automatized skill. For such skill, what could be the invariance the common or the constant upon which automatization developed? The constant could not be a specific ball speed at relative three dimension point (x, y, z) since a player can not quantitatively aware of such detail. Reasonably, the player could associate the skill to a ball attended to be with speed of some range and within a relatively positioned three dimensional space. So, the constant for the skill is likely to be a vaguely specified state space that is defined by the range of speeds and region of space above the head. It is invariance if the ball is categorized or recognized as a point in such state space. The cognitive or the "conscious" content, which is the basis of the automatization process, is invariant. The detailed spatial sensory information is communicated to the low-level motor action processors in the sensorimotor interaction pathway; this happens with or without the automatization effect. Eliciting "conscious" sensation takes about 500ms while most motor reaction times are shorter. That means the unconscious online control of action happens with low-level, wideband communication path in the sensorimotor modules. With implicit and explicit pathways for inter-codelet communication, our model suggests that despite many events, qualities, and other things that come to awareness there are many more, which are sensed and which in turn shape and effect motor responses but never make it to consciousness. There is evidence that there are anatomically distinct pathways for conscious and unconscious control of action. In vision system, there is growing evidence that an occipito-parietal (dorsal) route supports the "how" implicit processing and an accipito-temporal (ventral) route supports the "what" explicit processing (object identification) in directing action on a targeted object (Jeannerod & Rossetti 1993; Milner & Goodale 1995). In the above example task of heating a ball, while decision to heat the ball is a "conscious" process, the online motor control or continuous adjustment of the hand movement towards heating the ball is unconscious. Human time resolution for conscious processing does not suggest having such motor controls with the involvement of consciousness. Online motor control usually requires a sensory input with content passing via the dorsal pathway. In our model, such online control of action with relatively short reaction time is implemented by behavior streams and the underlying codelets (motor and expectation processors) and their dorsal pathway type of interaction with the perceptual module (underlying attention and information codelets). Such online motor controlling behavior streams (at the lower-end of goal hierarchies) as chunks are building blocks for high-level behavior streams that usually are performed by invocation of multiple "conscious" goal selection (Chapter 4). The automatization process as implicit learning process reduces the cognitive load by replacing the involvement of awareness in selecting actions with automatic response priming.

Studies by many researchers using masking techniques, lesion, and action-blindsight disorders suggest that there is disassociation between the neural pathways leading to visual awareness and visuomotor (implicit) processing. Our action selection model supports "consciously-mediated" action selection process – in the execution of a task, intermediate events in the environment, as consequences of actions under the control of an active behavior stream or dominant goal hierarchy, come to awareness. There are the "conscious' and the unconscious components in the process. While the "conscious" components are those that are reportable events (or broadcast content) in the task execution process, the unconscious components include the behavior stream (goal context hierarchy) and the different codelets that realize the motor mechanism.

To generalize, automatization is developed for a task relevant to a situation, which is attended to be invariant. A behavior codelet uses unconscious, detailed information passed by associated attention codelets to rebind its variables and as a result to tune its action. The direct interaction developed between behavior and attention codelets as a result of automatization creates an optimal task execution. We strongly suspect that interactions among low level processors based on their association will explain sensory-motor coordinations such as priming effects and reflex actions.

5.8 Test and Results

5.8.1 Domain Specification

We choose a simple domain for our test - to implement a behavior stream for a walking sequence. If we sample our walking, we can have snapshots of behaviors, which each represent the activity of specific muscle groups. The walking sequence could be described as: step one – move right foot up and ahead, step two – move right foot ahead and down, step three – move body weight to right foot, step four – swing left foot up and forward, etc. We can define the walking sequence with fine or coarse grained behaviors, which is a design decision. We specified a behavior stream that contains six step behaviors for a walking sequence, a behavior to observe the walk way, and a goal of "walk-forward." We also define a drive "like-walking" as an internal motivator for the behavior stream. In reality the 'need to walk' becomes active in service of larger goal hierarchies that satisfy deeper drives such as hunting and gathering. Each defined behavior may represent twitchings of smaller muscle groups. In the case of robots, each behavior may deal with actions that affect components with multiple freedom of movement. In our model, the twitchings are represented by behavior codelets, and we need to specify one or more codelets that underlie each behavior. Any domain task with a sequence of behaviors could be a good choice to test automatization. But, we choose this domain for its clearly defined steps that every body could understand.

Automatization is an incidental learning process and helps to internalize procedural skills via practice. As automatization gets stronger, performance improves and conscious control fades. We can consider that automatization is a type of chunking process. Our tests qualitatively show the improvement of performance and fading of "conscious" control with repetitive execution (practice) of a novel procedural task. Here we use a simulated human walking sequence task. Walking is an automatizable task with a sequence of behaviors that activates different muscle groups with high level of coordination to move our two legs and carry our body around without falling down. We normally develop skills for walking and other such motor skills at an early age. We also may need to relearn them if, and when, bodily injuries that affect those skills happen. Using the walking sequence task, our test will show that: (i) a "consciously" controlled novel procedural task helps to build automatization; (ii) "conscious" control of a task fades as automatization matures; and (iii) performance improves qualitatively as automatization matures.

The behavior network is designed; then we describe the specification in the XML file using the BNDL grammar. Figure 5-6 shows the description of the third walking step behavior. It has two precondition propositions: second-walk-step-done, and step-distance-ready. Each precondition proposition has a variable. The second-walk-step-done proposition has a variable (named *varSecondWalkStepOutput*) of type "StepSize," and its value represents the actual step size of horizontal movement. The step-distance-ready proposition has a variable (named *varStepDistance*) of type "FullWalkDistance," and its value represents the total distance that is expected to be covered by a complete sequence of the walk steps. This design suggests that each walk step can consider the distance covered by the previous walking step and the expected distance to be covered by all the walking steps. The add list has one proposition – "ThirdWalkStepDone," and the delete list one proposition – "SecondWalkStepDone."

Although not shown in figure 5-6, each variable has value and context (problem space) slots, which are bound to values during run time. The third walking step variable has only one behavior codelet (named BCDoThirdStep) under it, and this codelet reflects the complete specification of the behavior and implements the complete behavioral action. Only generic expectation codelets are used under all behaviors, and so none are explicitly specified in the XML file. All variable types and required variable instances are specified in the XML file.

```
<!-- 3rd step walking behavior is defined here -->
<behavior element-name="beh3rd-Walk-Step" mode="true" type="action">
    <alias> Third-Walk-Step </alias>
    <precondition element-name="SecondWalkStepDone" >
        <variable element-name="varSecondWalkStepOutput" type="StepSize" >
        </variable>
    </precondition>
    <precondition element-name="StepDistanceReady" >
         <variable element-name="varStepDistance" type="FullWalkDistance" >
        </variable>
    </precondition>
    <addition element-name="ThirdWalkStepDone" >
        <variable element-name="varThirdWalkStepOutput" type="StepSize" >
        </variable>
    </addition>
    <deletion element-name="SecondWalkStepDone" >
        <variable element-name="varSecondWalkStepOutput" type="StepSize" >
        </variable>
    </deletion>
    <bcodelet element-name="BCDoThirdStep"
              class-name="Automatization.Walking.BehaviorCodelets.BCDoThirdStep"
              relevant-info="" order-id="0" context="NUL" effort="1" >
        <precondition element-name="SecondWalkStepDone" >
            <variable element-name="varSecondWalkStepOutput" type="StepSize" >
            </variable>
        </precondition>
        <precondition element-name="StepDistanceReady" >
            <variable element-name="varStepDistance" type="FullWalkDistance" >
                    </variable>
        </precondition>
        <addition element-name="ThirdWalkStepDone" >
            <variable element-name="varThirdWalkStepOutput" type="StepSize" >
            </variable>
        </addition>
        <deletion element-name="SecondWalkStepDone" >
            <variable element-name="varSecondWalkStepOutput" >
            </variable>
         </deletion>
    </bcodelet>
</behavior>
```

Figure 5-6: An example BNDL program (XML format) that specifies a walking step behavior.

As discussed in chapter 4, once the BNDL programming is completed, the next task is to code the behavior codelets and the necessary attention codelets. In doing so, the behavior codelets inherit the base behavior codelets, and the attention codelets inherit the base attention codelets. For the domain of walking, we have three explicit attention codelets.

An attention codelet watches for readiness for another walk sequence, a second watches for walking range at front, and a third watches for whether there is a need for walking. The need for walking is a volitional intention that may come to being for the service of high-level goal structures such as going for a walk around the block.

We also need to implement the main agent program that starts all the necessary modules. In this test implementation, the first three steps of the cognitive cycle are represented by a stub. In the IDA model, the current situation is recognized by these three steps of the cognitive cycle (CC): (1) unconscious perception, (2) the storing of percepts in the preconscious buffer, and (3) automatic retrieval of local associations from transient episodic memory and long-term associative memory, using the contents of the preconscious buffer as a retrieval cue. The contents of the preconscious buffer and the retrieved local association define the current situation in the long-term working memory. The stub is a graphics interface where a user inputs the required information, which is stored in the long-term working memory. There are no need to start the perception, long-term associative memory and transient episodic memory modules. Only the "consciousness" and behavior network modules are started.

Once a software agent is started, an internal or environmental source of motivation plays a role to induce internal state change that triggers some action. The change comes about by the stub provided by the graphics interface, which make changes in the long-term working memory. The current situation, which is reflected in the working memory, includes the 'need to walk', 'walking range', and 'ready for another step' with the appropriate binding values. The agent deals with a problem space that the current situation presents. The three attention codelets, which find relevance in the current situation in the long-term working memory, compete for conscious access – the 4th step in the cognitive cycle. The specific situation detected by the winning attention codelet becomes the conscious content. The conscious content is broadcasted (5th step of the cognitive cycle) to recruit resources that handle the situation. Some behavior codelets, which find themselves relevant to the situation (broadcasted information), get recruited and jump to the side line (6th step of the cognitive cycle). Then they prime and instantiate their behavior streams – setting the goal context hierarchy (7th step of cognitive cycle). In the example, we have only one behavior stream to instantiate. The behavior codelets with a priming role bind variables of their associated behaviors with the relevant information they received via the broadcast; they also inject environmental activation energy to the instantiated behaviors they primed. In the BNManager, the activation/inhibition dynamics selects a behavior to be active (8th step of the cognitive cycle). The behavior and expectation codelets that are under the active behavior (or dominant goal context) are signaled (via the representative codelet) to jump to the playing field. In the playing filed, the behavior codelets execute their specialized actions (9th step of the cognitive cycle) and at the same time expectation codelets monitor actions of the behavior codelets and report the outcome. Expectation codelets play a role in bringing about conscious involvement to handle unexpected situations; whether it will be to deautomatize over-learned procedural skills or as we will discuss in chapter 6, to detect a non-routine problem situation.

We would like to point out that some cognitive cycles pass with no explicit action; this happens when the behavior network is "thinking" or the behavior selection has not matured yet. So, the binding of variables in the instantiated behavior stream may take multiple cognitive cycles.

5.8.2 Automatization Experiment

A practice with walking takes place each time (a trial) all the walking steps are executes in their natural order. Referring to equation 5.5, the parameters for logistical functions to build associations are set as: $A = 4.18$, $a = 1.4$, and $c = 2.5$. The walking process is set to run many times so that experience could develop. During the run of each trial, we collect data on the number of broadcasts and the time taken to complete the trial.

Figure 5-7 shows that the count of conscious broadcasts per trial reduces as the walking experience develops. This is because of the fact that the activation level of attention codelets decreases with experience (strong association with behavior codelets) and an attention codelet succeeds in getting access to "consciousness" with the probability of its normalized activation level. The result shows that "consciousness" fades as automatization develops.

Figure 5-8 shows our result on the effect of practice on performance. With practice, the response time or performance improves.

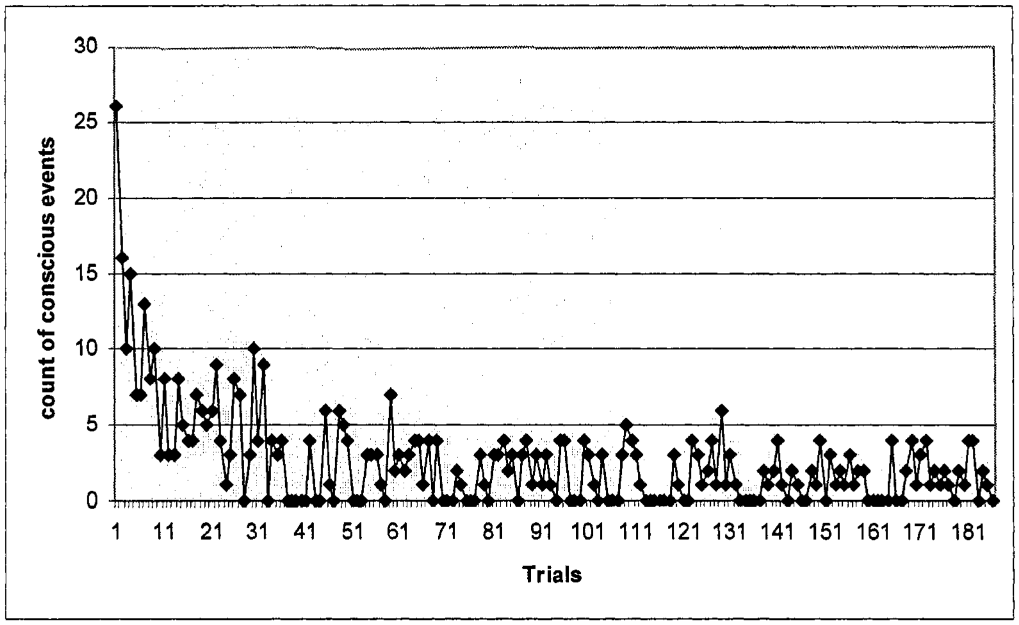

Figure 5-7: "Conscious" access fades with practice.

Automatization as an implicit or unintentional learning process embeds knowledge in the form of change of connection weights between coactive codelets; it brings optimal behaviors in performing well practiced tasks.

In our test, we measure the real time of completing each walk cycle. This has its own side effects such as the dependence of this data on the speed of a computer we used and the dependence on arbitrarily set delays at different modules of IDA to satisfy computational needs such as thread synchronizations. Qualitatively, as we shown in figure 5-3, automatization bypasses some cognitive steps of IDA including those considered expensive. We believe that a better measurement of the effect of practice on performance is possible if we can get a good estimate on the timing of each of the nine cognitive cycle steps of IDA. We hope to get the estimates in some future cognitive science and/or neuro sciences researches.

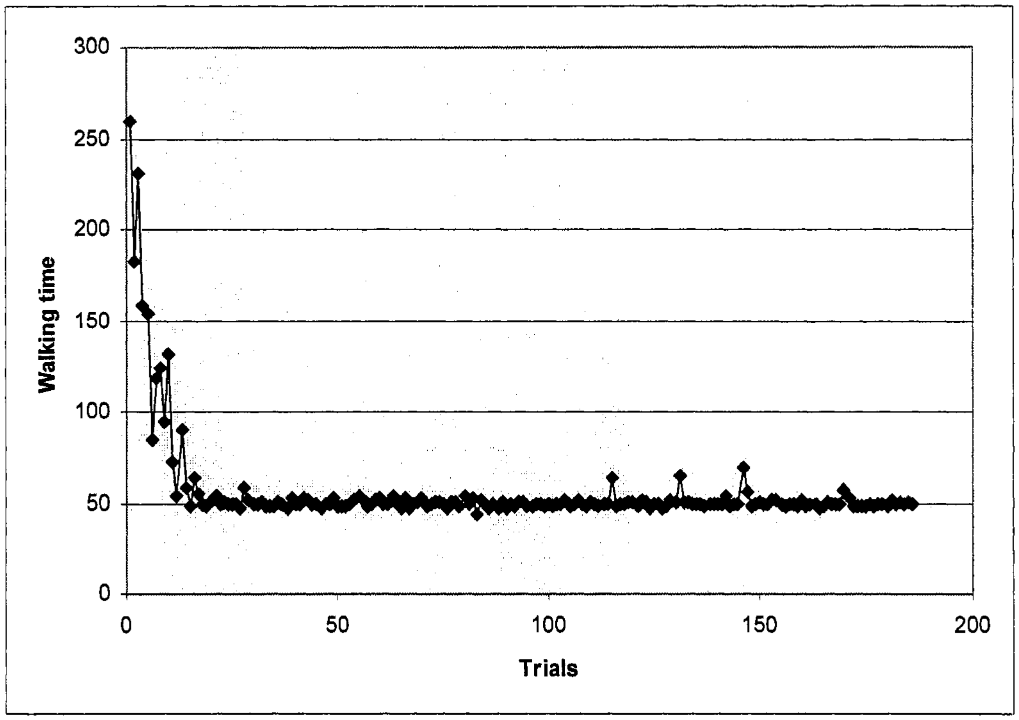

Figure 5-8: Performance improves with practice.

5.8.3 Deautomatization Experiment

To test deautomatization in our domain, we should be able to introduce error situation. We implement error situation by providing a stub in a graphics environment that disables and enables the proper functioning of behavior codelets that underlie one of the walking step behaviors. If a behavior codelet under execution could not fulfill its task, the sibling expectation codelet detects the failure of the behavioral action – thus a situation for deautomatization is created. After automatization is well developed (number of walking related broadcasts is near to zero), the deautomatization experiment is done by manually introducing an error situation and once the error situation is reported "consciously" (which is displayed on the graphics interface), we manually remove the error situation again using the provided graphics interface. Then we record the number of broadcasts needed to complete each walking cycle. Figure 5-9 shows that "conscious" monitoring is reinstated for a while to monitor the well practiced walking task.

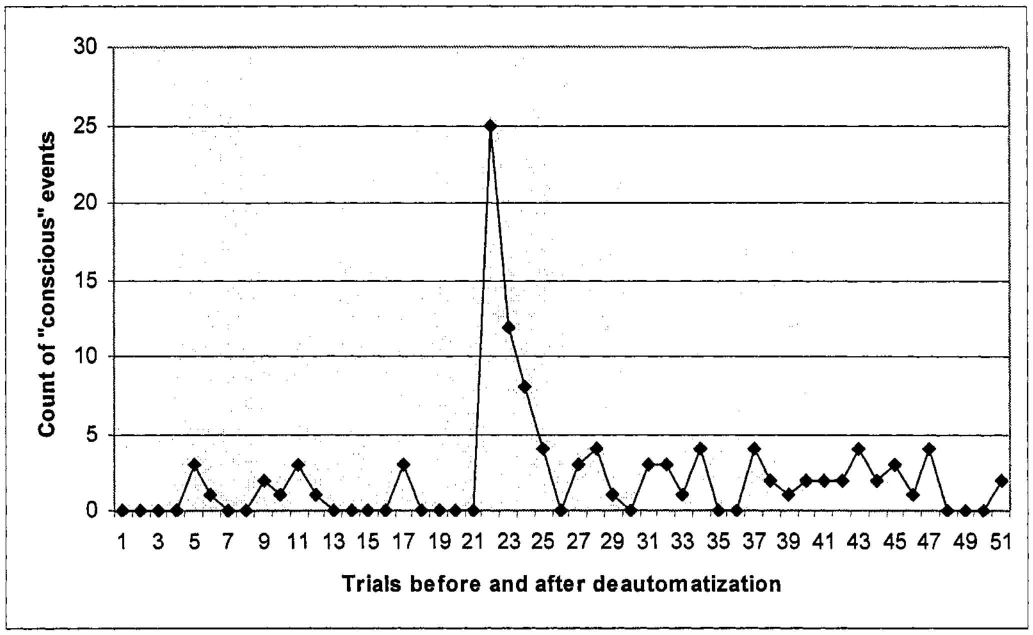

Figure 5-9: Deautomatization effect – reinstates "conscious" access for a while after failure situations.

Although, we can tune the decay rate of the deautomatization effect, we have not found human data on a possible decay rate or on how long a deautomatization effect stays. But, we speculate that the decay rate may depend on interference from competing tasks and the motivation level of avoiding the error situation.

In the plots shown in figures 5-7, and 5-9, events associated with matured automatized actions get "conscious" access. That is, despite strong automatization, "conscious" access is not completely faded to zero as shown by the ripples in the plot. This is due to two main but not independent reasons. (i) Automatization mechanism fades "conscious" access by decreasing activation levels of attention/expectation codelets; an activation

level of a codelet, which decide the codelet's probability of gaining access to "consciousness," will approach zero asymptotically but never be equal to zero; that is, strong automatization will not result an automatic denial of access to "consciousness."
(ii) Attention/expectation codelets compete for "conscious" access. The less competition a codelet encounters a higher probability to gain access to "consciousness;" in our testing setup, we did not introduce competing tasks to the automatized task; less competition for "conscious" access from attention/expectation codelets associated with competing tasks; absence of competing tasks will not change the nature of the curves (figures 5-7 & 5-9) but would simply reduce the number and/or frequency access to "consciousness" – less number and/or reduced amplitude of the ripples.

5.9 Conclusion

For humans, an automatized task is more efficient and can be performed more quickly by removing the need for the limited and serial resource of consciousness. In this chapter, we have discussed a mechanism for automatization and deautomatization for a software agent that implements Global Workspace Theory of "consciousness" (Baars, 1988). Automatization develops by building associations among relevant low-level processes that we call codelets. Strong intra-codelet association links allow low-level intra-codelet communication, which does not require "consciousness." As a task is automatized, attention is incrementally withdrawn by damping down the competitive strength of associated attention codelets. As the strength of an attention codelet decreases, so does its likelihood of being able to bring its information to consciousness. Therefore, a task that previously required conscious involvement at every step will not need such in the future.

In other words, automatization brings about lowering of cognitive load and optimal behavioral actions in executing regular tasks. Still, for most cases, even an automatized task requires a conscious event or a voluntary action to recruit it into execution (Franklin, 2000; Kondadadi & Franklin, 2001). Also, "consciousness" may be needed at various points during the task's execution when information is needed to bind variables in behaviors and their codelets; this occurs when any condition for automatization (invariance of situations) violated in executing a task.

An automatized action fails when it does not produce the expected outcome. Expectation codelets associated with each action detect the outcome and try to bring the outcome information to consciousness. The likelihood that the expectation codelet will get to consciousness (its strength) is dependent upon the degree to which the expected outcome differs from the observed outcome. If the outcome is an unexpected state and the expectation codelet succeeds in getting to "consciousness," then deautomatization takes place to reinstate conscious control. We showed how the deautomatization cognitive function is modeled to break apart the automated skill into consciously accessible components so as to bring conscious control into effect and consequently to modify/edit the skill explicitly. For this to happen (1) the automatization of the skill is disabled. The expectation codelet passes the relevant signals to its sibling behavior codelets, and from them the error status signals propagate to other strongly associated codelets; (2) skill adaptation is triggered. Expectation codelets that encounter errors always try to bring the error information to "consciousness."

Our cognitive model and the mechanism presented in this chapter implement a qualitative cognitive model that is consistent, primarily, with Global Workspace Theory, but also take into account findings in other areas of cognitive research. Automatization has been shown to have some level of dependency on attention (Kahneman & Chajczyk, 1983) and expectation (Logan, 1980) processes. Our attention and expectation codelets correspond to these roles and perform the necessary functions dictated. Our mechanism agrees with the view that automatization is a multistep algorithmic process as described by Schneider and Shiffrin (1977), rather than a single step memory instance retrieval of past experience as suggested by Logan (1998). Our mechanism postulates that this multistep algorithmic process is underlain by associative weights among low level processes and the implicit adaptation takes place as a result of changes in the associative weights. The deautomatization process breaks the automatic skills to individual components by disabling the effects of automatization, which mostly occurs at the behaviors' level. The presented deautomatization algorithm suggests that automatization cannot happen as single step instance retrieval from memory.

The mechanism described in this paper details a computational algorithm that produces the functional attributes of automatization. In addition, it is shown how this mechanism qualitatively exhibits many of the human properties associated with the practiced/repeated execution of a skill. The underlying assumption is that what has been described here as a computational mechanism is implemented in humans neuronally.

6 Non-Routine Problem Solving

6.1 Introduction

We humans have the ability to devise unexpected, and often clever, solutions to problems we've never before encountered. Sometimes they are even creative. This ability to solve non-routine problems has played a central role in the development of our sciences and technologies. It would be useful to replicate this ability in software agents and robots, both for its practical value and for the light it would shed on human problem solving. On the practical side, agent architecture capable of non-routine problem solving would allow software agents and/or robots to deal more intelligently with highly complex and dynamically changing environments, and to cope with situations unforeseen by their designers. Applications might be to unmanned vehicles for exploratory or military uses, as well as industrial control systems. The tasks of many human information agents could be automated (Franklin, 2001). On the science side, each design decision taken in pursuit of such an agent architecture translates immediately into a hypothesis, hopefully testable, for cognitive scientists and neuroscientists (Franklin, 1997, 2000b; Franklin & Graesser, 2001; Baars & Franklin, 2003; Franklin, Baars, Ramamurthy, & Ventura, 2005).

Thus there is ample reason to pursue such architecture. But can such pursuits succeed? We argue here that there is a good chance that it can. The major source of our optimism is the software agent technology of IDA (chapter 2) to which this research contributes.

IDA incorporates much of what seems to be necessary for non-routine problem solving. What's needed is to add additional capabilities to its architecture.

In pursuing such a non-routine problem solving architecture, we'll need help from many sources. Psychologist Arthur Glenberg refers to this process of finding unexpected and sometimes clever solutions to non-routine problems as *meshing* (as cited in; Glenberg & Robertson, 2000). Meshing is typically accomplished in humans by putting together bits and pieces of knowledge and techniques that have been stored and, perhaps, used in the past to help solve other problems. Finding a solution often begins with identifying these bits and pieces, then typically continuing to find ways to mesh them. We will use these observations as a guideline. Consciousness provides the means to extensively mobilize the unconscious bits and pieces of knowledge that could contribute towards constructing a solution.

6.2 The Problem and Preliminary Ideas for a Mechanism

Although its "consciousness" module is designed to deal intelligently with novel, unexpected, and problematic situations, IDA is currently expected to handle only novel instances of routine situations. One message from a sailor asking that a job be found is much like another in content, even when received in natural language with no agreed upon protocol. Similarly, finding a new billet for one sailor will generally require much the same process as for another. Even the negotiation process between IDA and a sailor promises to most frequently be a relatively routine process.

However, we expect IDA to occasionally receive messages outside this expected group. For example, a sailor could request consecutive shore assignments as an alternative to a hardship discharge because of illness of a family member. Or a command could request an emergency replacement for an injured sailor in a critical position when the ship is due to leave port. Currently, due to the rarity and sensitivity of these issues, IDA refers these to a human detailer. Even so, *could* IDA handle such situations intelligently by virtue of its "consciousness" mechanism alone? We don't think so. Additional mechanisms will be required.

There are two main issues to be considered. The first involves the recognition of a novel situation and the relevant information contained in the message. While this is, indeed, an important and challenging issue, it is not the sense of nonroutine problem solving that we are addressing here. We consider this first issue one of perceptual learning, rather than nonroutine problem solving. The second issue is to determine how to respond to the situation in an appropriate manner. This dynamic adaptation of procedural knowledge will be addressed here as the sense of nonroutine problem solving to be undertaken.

In the current implementation of IDA, if the situation is truly novel, rather than routine with novel content, IDA's slipnet may have no idea-type node corresponding to the incoming message. What will probably be needed is a "novel content" node that will allow relevant recognized items from the message to be written to the workspace, even though the idea type isn't recognized. These items would be the first building blocks of a solution. How does the novel content node know which items are relevant? The answer, of course, is that it does not. Instead, it would assume that all recognized information is potentially relevant. Keep in mind that novel information may be fairly common in messages; it is only when no other idea type is recognized that the novel content node becomes prevalent enough to be acted upon.

Once the information gleaned from the message is written to the workspace; there may not be an attention codelet that will respond to it. This indicates that non-routine problem solving will require specific attention codelets whose task is to respond to unknown situations. But how is such a codelet to recognize such situations without knowing about all the usual occurrences? It seems to be similar to the problem faced by the immune system and may well require the same kind of solution. Just as a human would, IDA learns new procedural methods incrementally based on related, known concepts. This is more commonly described as "learning at the fringe" (Doignon & Falmagne, 1985). For example, let's say that IDA receives a message saying that a sailor needs to stay on shore duty for his next rotation because his wife has been diagnosed with cancer. If he can't get another shore duty, then the sailor will have to take a hardship discharge. In cases like this, the Navy is likely to waive the adherence to the usual sea-shore rotation and allow the sailor to stay on shore duty. IDA is not designed to handle this situation because it would require disregarding a Navy policy. However, IDA may have a behavior stream that handles a different situation where it must ask a human whether she can disregard a different policy. Let's step through how "conscious" nonroutine problem solving would handle this example.

First, as described above, all the information that IDA can glean from the incoming message is written into the workspace. A nonroutine problem solving attention codelet notices that no other attention codelet is attempting to deal with this new message and decides that it needs to bring the information to "consciousness." Assuming that it wins the competition for consciousness, it will broadcast the information known about the situation based on the message.

While there are no behavior codelets that directly respond to the information broadcast, various behavior codelets that recognize pieces of the information will respond to varying degrees, depending on how much of the information they are familiar with. These initial respondents will become the base from which possible solutions will be generated. In this way the search space is constrained to a pool of likely candidates. In addition, their activation levels at the time, corresponding to each behavior codelet's assessment of its applicability, are used to further bias the search for a solution.

The behavior stream associated with the behavior codelet that responded with the highest overlap is analyzed by a special nonroutine behavior stream to determine the differences in the information at hand vs. the information that the stream needs. Based on this analysis, individual behaviors that are not appropriate for the current information contained in the incoming message can be isolated. These individual behaviors are then altered to accommodate the most similar available information. In our example situation, this would likely mean that the behavior at the beginning of a sequence that asks a human detailer about disregarding a different Navy policy would be isolated. In this case the most naive and obvious alteration happens to be correct; namely, replacing the old policy type with the policy type in question for this situation. The same modification must be made in all behaviors in the stream where the old policy type was used.

Wherever modification is necessary, a new behavior is created and attached to the existing stream. This process repeats until all information needed by the original stream is supplied by available information through modifications of behaviors. If the available information is not sufficient to adequately alter the current behavior stream, then the process starts over with the next most likely candidate. At the end of this process there will be a new stream available for instantiation that will attempt to deal with the nonroutine problem.

Obviously, it is not known whether the newly generated solution will actually solve the problem at hand. For this reason, every new stream has what might be called a "probationary period." After using the new stream to deal with the nonroutine problem, IDA waits to see what response is produced. If the response is positive, then IDA concludes that the new stream was successful in dealing with the given problem. Otherwise, IDA throws out the newly created behaviors and attempts to create a new behavior stream, just as described above, with the addition of any information gathered from the negative response.

In addition to attention codelets that look for situations where no other attention codelets have responded within a reasonable time, there are other situations that may cause nonroutine problem solving to occur. Other nonroutine attention codelets may respond to an internal oscillating state that is not making progress toward the currently active goal context. Finally, with every action that is taken, there is an expectation of an externally or internally perceivable result, indicating that the action was successful. If this expectation is not met, then an expectation codelet, a particular type of attention codelet, will attempt to bring this unexpected state to "consciousness;" this is also a situation when a non-routine problem must be addressed. Detecting internal oscillation and other nonroutine impasse situations are a type of metacognitive processes that monitor what is happening in the goal context system, and to which attention codelets can respond. Solving non-routine problems often requires several attempts.

For IDA to accomplish the tasks just described, mechanisms for combining or modifying individual behaviors, even individual behavior codelets, will be needed. Also, dealing with non-routine situations may well require composing a message for which no suitable scripts are available. Modification and/or combinations of existing scripts will be required. Again, a mechanism for deciding which existing scripts to work with is presumed ("consciousness" once again).

Also, we cannot expect this to be a one-shot process. Sometimes several tries will be required to build a solution. Again, a new mechanism will be needed to handle this doubling back. And how is IDA to decide when a solution is acceptable? Although we've come up with some possibilities, such as described above, there is still much basic research to be done here, but the handling of non-routine problems does seem doable by an extension of the IDA architecture.

Following global workspace theory, we conjecture that creative solving of non-routine problems by humans is accomplished using consciousness in the role of a recruiter of relevant resources. We further conjecture that such creative problem solving also requires a meshing mechanism capable of cutting and pasting various goal contexts along with their associated processors. Finally, we conjecture that some sort of mechanism (processor(s)) is needed for recognizing such problems and training attention on them.

Translating all this into computational terms, we conjecture that IDA could be provided with attention codelets that recognize non-routine problems and bring them to "consciousness" and with a sufficiently capable cut and paste mechanism for behavior streams and scripts, so that it will be capable of providing clever, possibly creative, solutions to non-routine problems from its domain.

In general, IDA's perception and the memory modules may need to be involved in the recognition of nonroutine situation and in the process of constructing creative solutions. But the scope of the research here is limited to the integration of non-routine problem solving mechanisms with the action selection and "consciousness" modules. In the coming sections of this chapter we will present the details of a mechanism that could build a novel solution to a problem, which is recognized to be nonroutine in a given situation.

6.3 Approach and Design

6.3.1 Routine and Non-Routine Problems

Routine problems are the ones that have readily available *applicable solutions*. Non-routine problems are novel problem situations that arise from *unavailability of routine solutions* that can handle them. More specifically, non-routine problems cannot be categorized as any of the routine classes of problems, and thus none of the available behavior streams could be applicable.

In general, a problem solving process may include the capabilities of noting differences, similarities and, to some extent, analogies between a non-routine problem and well-established routine problems, and the resulting information is used to construct new solutions by modifying existing solutions. A mechanism may need to breakdown complex structures of procedural knowledge to lower-level components, then using the available knowledge to construct new goal structures as solutions. The breakdown of structures can continue to the simplest or collection of base units that Edelman (1987) called primary repertoires. Problem solving, whether routine or non-routine and like traditional AI planning, is always specified with goal and the initial states.

6.3.2 Detecting Non-Routine Problems

In IDA, there are two scenarios in which a non-routine problem can arise. 1) If IDA encounters an unexpected type of situation that could not exactly be categorized in any one of the known classes of problems; it is a non-routine problem since there will not be known behavior streams to handle the situation. 2) In an active behavior stream (dominant goal hierarchy), an executing behavior (dominant goal context) may fail to achieve its expected outcome, which happens if IDA encounters perturbation in its environment, a variation in its domain, or if any one of the acting behavior codelets malfunctions. This situation crops up if there is no alternative routine solution that corrects the error situation. In both scenarios, the non-routine problems arise from instances of failed expectations.

IDA perceives novel incoming messages all the time; how do we detect non-routine problems in the first scenario above? If a novel message instance has a likeness to one of the recognizable idea-types, then it is a routine problem situation since IDA has a readily applicable solution or behavior stream. On the other hand, if a novel message instance is different from all the known idea-types, then there will not be an applicable solution and thus a non-routine problem arises. The non-routine problem is specified with one or more of its goals, its environmental conditions and internal state. A special attention codelet will monitor the workspace for non-routine problems; when it finds one, it activates itself and competes to bring the non-routine problem description to "consciousness."

How does IDA detect non-routine problems that arise from execution errors in a running behavior stream? Each behavior, besides behavior codelets, has one or more expectation codelets under it. When a behavior starts execution and dispatches its behavior codelets for action, at the same time it activates each of its expectation codelets. The role of an expectation codelet includes watching, evaluating and, perhaps, reporting on the outcome of the action. The function of watching events that are associated with the action of its behavior makes an expectation codelet a particular kind of attention codelet. The monitored changes resulting from a behavioral action are characterized and formulated as an outcome. If the actual outcome is different from the expected outcome, the expectation codelet detects an error situation. The execution error, as a non-routine problem, is described as an unfulfilled part of the outcome (goal), the fulfilled part of the outcome (state to be protected) if there is any, and the contextual state at the point of failure. The contextual state includes the perceptual context from working memory, long-term memory and the internal state in the behavior net. The expectation codelet that detects an execution error joins the playing field and competes to bring the error information to "consciousness."

6.3.3 Constructing Solutions for Non-Routine Problems

Solving non-routine problems is a deliberative process that needs multiple cognitive cycles. The solutions require the creation of new goal structures or behavior streams. We propose a high-level, but detailed, design of a Non-Routine Problem Solving (NRPS) module for IDA. The NRPS mechanism should be a consciously mediated reasoning system that breaks down and combines procedural knowledge pieces to produce new behavior streams. An AI planner is a type of problem solver and a deliberative action selection system. But, in our system deliberative action requires "consciousness," which is not assumed in AI planners. Particularly, we propose a non-routine problem solving module of IDA as a special planning (regressive and dynamic) behavior stream, called NRPS behavior stream. As any other behavior stream, the NRPS behavior stream gets instantiated and executed in multiple cognitive cycles. But its particular task is to generate a novel behavior stream as a solution for a given non-routine problem situation.

We would like to point out two driving principles for our approach in designing a non-routine problem solving mechanism. The first is that complex processes include simple and primitive processes. The complexity arises from the structure that governs the interaction of the primitive processes in time and space. The second is that creativity is an important feature in handling non-routine problems; we believe that creativity is the process of using simple units to build larger and more complex structures such as behaviors and behavior stream templates (procedural schema). An important capability of non-routine problem solving is to create new behavior codelets using refined domain knowledge, but this level of creativity needs further investigation and we do not address it here. That is, we limit the primary repertoire to the level of codelets, which can be considered as neuronal groups (Edelman, 1987).

An attention or expectation codelet that detected a non-routine problem may win the competition to "consciousness," and the problem information content is broadcast. Then behavior codelets find themselves relevant to the pieces of the broadcast information and instantiate the associated behavior streams. Particularly, there is a special non-routine problem solving (NRPS) behavior codelet that becomes relevant to all broadcasts related to all non-routine problems. The NRPS behavior codelet instantiates the NRPS behavior stream. Then the instantiated NRPS behavior stream starts to play its role with the help of "consciousness", i.e., whenever possible it generates a behavior stream template that may be a solution for the non-routine problem in consideration.

Non-routine situations should be recognized and brought to “consciousness” by nonroutine attention codelets. Our approach to managing nonroutine problems is functionally equivalent to what Shallice (1988) call a Supervisory Attention System (SAS), which is associated with the prefrontal cortex. SAS could correct errors and determine novel action plans, and it has a number of functions (Shallice, 1988). (i) SAS Monitor: This can be considered as a nonroutine situation detection system. As is the case in IDA, SAS Monitor is connected to the long-term working memory. Novelty detection, which may take multiple cognitive cycles, notices departures from what is expected while looking at long-term working memory (LTWM) (Ericsson & Kintsch, 1995). Looking is done by cuing the LTWM with the content of current percepts and retrieved content from different memories, which in turn are cued by the current situation. In IDA, this role should be addressed in the perception and the different memory modules; (ii) SAS Modulator: The SAS must activate relevant actions and goal structures while attenuating irrelevant actions, particularly to try out available solutions to the novel situation. In IDA, this functionality is realized by behavior codelets that respond to pieces of information from nonroutine problem handling related broadcasts and which instantiate relevant behavior streams. This is similar to adaptation by assimilation (Piaget, 1983) in the sense of using existing solutions to handle novel situations; (iii) SAS Generator: SAS is used to generate novel strategies (such as altering goal structures or action plans) to solve nonroutine problems. Our research contribution in this chapter is to present a design for a mechanism that realizes the function of the SAS generator.

6.3.3.1 NRPS Behavior Stream

To incorporate a non-routine problem solving mechanism as a special goal structure, we add two constraints to the action selection mechanism discussed in chapter 4. (i) Each behavior codelet (primitive action) is associated with an expectation codelet. (ii) Based on the broadcast content, a recruited behavior codelet instantiates itself and relevant behavior streams in one of the two modes: (a) Execution mode – so that the instantiations become part of the action selection dynamics of an agent; that is, when a behavior codelet executes, its associated expectation codelet knows how to monitor the execution effects and how to report the outcome. (b) Non-routine problem solving mode – instantiations become part of the pool of actions that is available for the solution searching process.

The first constraint allows the NRPS behavior stream be able to cut and paste behavior codelets and coupled expectation codelets as atomic units easily. A behavior codelet and its corresponding expectation codelet are reorganized under different behavior as a unit during the solution search process. In the current computational IDA, such association is simplified by the implementation of a generalized expectation codelet module that could monitor actions of instantiated, sibling behavior codelets without explicit association. The general expectation codelet is constructed at run time by using a shared data-structure, which contains the bound precondition list, add list, and delete list, with sibling behavior codelets.

The NRPS behavior stream is shown in figure 6-1. The NRPS mechanism is based on a partial-order planning (POP) system (Sacerdoti 1977; Tate 1990; Wilkins 1990) - and the service of "consciousness" to recruit relevant resources in its solution searching process.

Variations of partial-order planners allow rich domain knowledge representation and are used in real world domains.

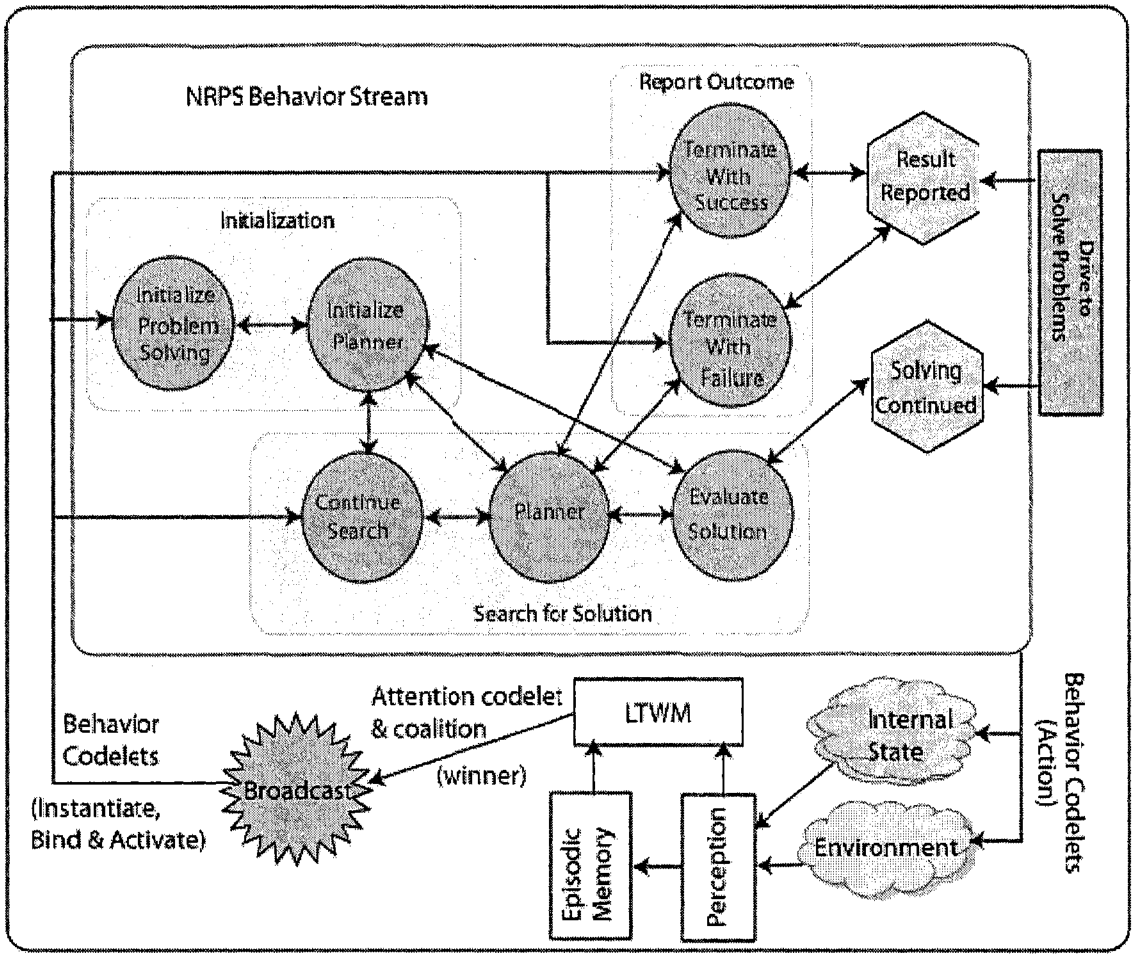

Figure 6-1: Non-routine problem solving (NRPS) module is a special behavior stream (goal-context hierarchy), which guides a deliberative process for problem solving over multiple cognitive cycles.

O-Plan (Tate, Drabble, & Kirby, 1994) and Sipe-2 (Wilkins, 1990; Wilkins et al., 1995) are domain independent partial-order planners that have been used in a number of real world domains. Such planners allow plan repairing and interaction with humans, which are features particularly important in IDA's domain. Erol, Handler, and Nau (1994) presented a formal definition of partial-order planning and proved that it is sound and complete. Wilkins (2001) discussed the comparative advantages of partial-order planners.

Standard POP systems usually take initial state, goal state, and all available actions of an agent and return a plan if one is found. For our NRPS mechanism, the POP system should be modified so that it could deal with a limited set of actions to start planning and to allow "conscious" deliberation if an impasse or a contentious point is reached in the planning process. An impasse is reached if a planner cannot find an action (in the current set of actions) that could satisfy an open subgoal, which we call an impasse subgoal (or contentious subgoal).

The NRPS behavior stream uses a modified POP system (the algorithm is shown in Appendix C-2) that takes a starting-plan, current-pool-of-actions or possible steps and a possible impasse subgoal and then returns resulting-plan, impasse-subgoal, and result-explanation, which provides human readable information on the result. A "consciously" mediated planning process continues until a consistent plan is obtained as a solution (no impasse-subgoal) or no solution is found (the impasse-subgoal exists and no new relevant action could clear the impasse-subgoal.) Next, we will discuss the role of each behavior in the NRPS behavior stream (fig. 6-1); its specification is given in appendix C-1.

i. Initiate Problem Solving

This behavior is the one that sets the problem-solving context. Its variables are bound by the underlying NRPS behavior codelet that responds to a broadcasted non-routine problem description (initial state and goal state). That is the NRPS behavior codelet instantiates the NRPS behavior stream. Also, relevant behavior codelets respond to the broadcast of the same non-routine problem description, as the behavior streams they instantiate and bind form the initial set of actions available to the planner. When this behavior executes, it asserts the readiness of the initial state, goal state, and the initial actions to the other behaviors in the NRPS behavior stream. As such, it sets the context for solving the problem in hand.

ii. Initialize Planner

Using the problem context set by the *Initiate Problem Solving* behavior, this behavior compiles the planner initialization parameters. Particularly, on its execution it does the following: (i) Using the initial and goal states, it creates the initial empty plan, in which the *Start* dummy action has literals in the initial state of the problem as its effects, and the *Finish* dummy action has literals in the goal state of the problem as its preconditions. (ii) Nondeterministically, it chooses an open subgoal (one of the literals from the goal state of the problem) and sets it as an impasse or contentious subgoal, which may or may not be an actual impasse but needs to be set for the proper initialization of the planner. (iii) It initializes an empty list, which is used to store "consciously" deliberated impasse subgoals. Each detected impasse subgoal is registered in this list.

iii. Planner

This behavior executes to build a new plan based on the previous plan and the current pool of actions. At the end of the planning process, this behavior asserts an impasse subgoal (set to null if there is none), result-explanation, and the new plan. The planning algorithm is given in appendix C-2.

iv. Evaluate Solution

This behavior evaluates the solution search state. Particularly, it checks whether the search for a solution should continue, whether a solution has been found, or if the search has failed to find a solution. The checking process is simple and based on the asserted value of the impasse subgoal and whether this impasse has been deliberated before or not. If no impasse subgoal left to be resolved, then the evaluation is a *success in finding a solution* with the current plan being the solution to the problem. If an impasse subgoal exists and is in the list of deliberated impasse subgoals, then it has been dealt before and therefore the evaluation is a *failure to find a solution* to the problem. The result-explanation gives human readable information feedback to a human on how the failure arose. If an impasse subgoal exists, but is not in the list of deliberated impasse subgoals, then the evaluation is to *continue the search for a solution.*

The actions of this behavior's behavior codelets produce internal state change that asserts one of the three evaluation statuses (*success in finding a solution, continue the search for a solution, and failure to find a solution*) and possibly the impasse subgoal. After the internal state change is perceived, the evaluation content as a percept and whatever it cues from memory is copied to the agent's long-term working memory (LTWM). Relevant attention codelets, which watch for the solution evaluation in long-term working memory, try to bring the information to "consciousness." Upon winning the competition, the content of the attention codelet and its coalition is broadcast. Relevant NRPS related behavior codelets respond to the broadcast and bind the evaluation content to the appropriate behaviors in the NRPS behavior stream. If an impasse subgoal exists, its broadcast causes behavior codelets, with the subgoal in their effect list, to respond, and they become the new additions to the set of actions available to the *Planner* behavior for its next cycle of search for a solution. That is, "consciousness" is used as filter in the solution searching process and this is, we think, the main advantage of this mechanism over other planning systems.

v. Continue Search

This behavior is underlain by a behavior codelet that responds to a broadcast of an evaluation that asserts *continue the search for solution,* which has been written in the long-term working memory as a result of the action of "Evaluate Solution" behavior. After its variables are bound by its behavior codelet and upon its execution, this behavior compiles a new set of actions, which consists of the union of the existing actions and the newly arrived actions. This behavior asserts the preconditions that allow the planner behavior to continue the search with the possible addition of newly recruited actions.

vi. Terminate with Failure

This behavior is underlain by a behavior codelet that responds to a broadcast of an evaluation with content of *failure to find solution* to the problem. After its variables are bound by its behavior codelet, the behavior executes to compose an extended explanation on the failure situation and to terminate or give up on the solution search process for the problem. The failure explanation could help as input for human experts so that they can provide additional knowledge to the agent. Although it is beyond the scope of our discussion here, actual explanation composition could be done by another behavior stream.

vii. Terminate with Success

This behavior is underlain by a behavior codelet that responds to a broadcast of an evaluation with the content of *success in finding a solution* to the problem. The last plan produced by the planner is a solution. Upon its execution, this behavior converts the plan into a corresponding behavior stream template. The conversion happens based on two facts. (a) The solution as a complete plan is a total order of actions (behavior codelets and associated expectation codelets that can be applied sequentially to an initial state to produce a goal state. (b) The solution as a totally ordered plan could be specified by a set of partially ordered complete plans (behavior streams). For the current version of IDA's action selection mechanism (ASM), the conversion also should satisfy a critical criterion - behavior codelets that underlie a behavior should be independent or should be able to execute in parallel or in any order without affecting each other. In planning terms, two actions are independent of each other; if any change of order happens between the actions, the plan remains consistent – no cycle is introduced in the ordering constraints of the plan and no conflicts arise in any one of the causal links of the plan. We intend to allow future versions of the ASM to allow behavior codelets under a behavior to be able to execute in parallel as well as in temporal orders, and the above criterion will be modified accordingly.

The simplest conversion from the plan solution to a behavior stream template is to have each action (behavior codelet and expectation codelet) underlie a behavior. Such is a correct solution and may be the only valid way to have a conversion, but it would be the worst case scenario particularly in execution efficiency. We suggest the following heuristics for a better conversion to be applied in that order whenever doing so keeps the partial plan consistent: (a) Map a set of behavior codelets to an existing behavior. (b) Replace independently executable behavior codelets in the plan solution with a new behavior, and to do so by limiting the maximum number of behavior codelets that can underlie new behaviors. (c) Each action (behavior codelet and coupled expectation codelet) that is not converted in the previous two heuristics is mapped to underlie a behavior by itself. Reusing knowledge pieces is an important feature of agents; the first heuristics allows the reusing of existing structures in the behavior network. Having multiple behavior codelets that underlie a single behavior improves performance since the selection of a behavior in the behavior network dynamics leads to the parallel execution of the underlying codelets.

The actual solution to the non-routine problem at hand is a novel behavior stream template with the possibility of a new goal structure built using only existing behaviors, or only new behaviors, or a mix of new and old behaviors. The *Terminate with Success* behavior stores the new behavior stream template in the database of behavior stream templates or schemas and asserts the availability of a solution to non-routine problems by writing the success information in the agent's long-term working memory.

6.3.3.2 NPRS Behavior Stream at Work

We have discussed details of the role of each behavior in the NRPS behavior stream in the solution search process for a given non-routine problem. Here, we point out the flow of information processing in the instantiation and execution of the NRPS behavior stream.

To start with, a non-routine problem situation is perceived and associated with what could be remembered; then the problem description is eventually stored in the long-term working memory. IDA does this in its first three steps of its cognitive cycles (CC-1 to CC-3). An attention codelet that watches for a non-routine problem situation in the long-term working memory becomes active, with its coalition of information codelets, and the coalition competes for access to "consciousness" (CC-4). If and when the competition is won, the coalition gains access to the global workspace and the problem description is broadcast (CC-5). The NRPS behavior codelet responds to the broadcast of the non-routine problem description. Other behavior codelets may find themselves relevant to pieces of information in the same broadcast and can recruit resources that could help to construct a solution for the problem (CC-6). The responding behavior codelets bind their variables with the relevant information in the broadcast. The NRPS behavior codelet instantiates the NRPS behavior stream, binds the variables, and increases activation of the behaviors it underlies. The other recruited behavior codelets (in planning mode) with the behavior streams they prime become part of the initial set of actions for the problem solving process (CC-7).

Once the NRPS behavior stream is instantiated, it is part of the dynamics in the behavior network; then its behaviors are chosen (CC-8) and executed (CC-9) one at a time and over multiple cognitive cycles. The NRPS behavior stream guides the search for a solution to the problem with the help of "conscious" mediation when an impasse arises. The *Initiate Problem Solving* and the *Initialize Planner* behaviors are executed only to initialize the problem solving process. While the *Planner* behavior has the task of planning in the solution search process, the *Evaluate Solution* behavior has the task of evaluating the output of the *Planner* behavior and store the result in the long-term working memory. The standard “conscious” mediation process feeds back the evaluation content and possibly recruits additional resources (actions).

If the evaluation is to continue the problem solving process, a relevant behavior codelet binds (or rebinds) and activates the *Continue Search* behavior, which has the task of setting up and asserting the parameters required by the *Planner* behavior. As long as the evaluation is to continue the search and it being the dominant goal context (with one of its behaviors is a winner in the behavior net dynamics), the NRPS behavior stream executes in a cycle: execution of *Continue Search* behavior is followed by execution of *Planner* behavior, followed by execution of *Evaluate Solution* behavior, followed by “conscious” mediation, followed again by execution of *Continue Search* behavior.

The NRPS behavior stream terminates with failure or success. To do so, a “consciously” mediated result processing, which involves all or most of the steps of IDA’s cognitive cycle, takes place. The process involves many codelets that are particularly designed to the nonroutine problem solving task. Besides behavior codelets that underlie the NRPS behavior stream, there are many attention and information codelets with the role of watching for or recognizing LTWM content that relates to nonroutine problem solving. Among those are attention and information codelets that watch for a result status of failure or a success in the solution search process. If a “consciously” mediated evaluation content asserts failure in finding a solution to the problem the result reporting action is executed by the *Terminate with Failure* behavior with the task of writing an explanation of the failure situation. If the "consciously" mediated evaluation content asserts success in finding a solution to the problem, the result reporting action is executed by the *Terminate with Success* behavior with the task of saving the solution as a behavior stream template in the behavior stream template or schema database or procedural memory and writing the availability of a solution to the stated problem in the long-term working memory. The termination state written in the long-term working memory could be brought to "consciousness" by relevant attention codelets. Consequently, failure states could prompt the agent to explain the problem to humans and success states will allow the agent to apply the new solution.

6.4 Related Works

Nonroutine problem solving is the process of devising appropriate stream of actions as a response to novel or unexpected situations. Devising solutions is a type of learning or adaptation, which requires paying focus of attention or involvement of "consciousness" in handling the situation (Baars, 1997). Norman and Shallice (1986) and Shallice (1988) have proposed a functional model for a control of both routine (automatic) and non-routine behavior. This functional model is broadly equivalent to IDA's decision making mechanism. In the Shallice's mechanism, non-routine problem is an attention-based learning managed by a mechanism called Supervisory Attention System (SAS), which is mapped to the prefrontal cortex (Shallice, 1988). Focusing on the deliberate action of the mechanism, working memory provides the attention system to access the current intentions, goals, and relevant past episodes. The SAS enables error correction, troubleshooting, handling non-routine problems using supervisory sub-functions such as SAS Monitor, SAS Modulator, and SAS Generator. The *SAS Monitor* is a non-routine problem situation or novelty detector. It is primarily connected to working memory. Non-routine situation detection is departure or mismatch of known contents from what is perceived in the world, intended action, outcome of executed action. These are related to what is stored in episodic, procedural, and working memory pieces. The monitoring sub-function is a trigger mechanism to the other supervisory sub-functions. Although we do not address the function of the SAS Monitor here, we think that this function in IDA is connected to internal perception and the different memory types the agent may have. The *SAS Modulator* function attenuates the activation of inappropriate actions and enhances the activation of alternate but attended goal structures. Shallice suggests few possible modulatory responses which we do not cover here. The SAS Modulator function is a type of applying existing solutions or behavior streams to new problems – which is adaptation by assimilation (Piaget, 1983). The *SAS Generator* is triggered if the modulatory response by the SAS Modulator does not succeed; which asserts the unavailability of recognized strategy to the situation in hand. The SAS Generator must be able to produce a novel strategy (new goal hierarchy or partial plan) that could handle the novel or non-routine problem situation.

As the case of Global Workspace Theory, the SAS framework suggests learning is triggered and happens with the investment of attentional resources or involvement of "consciousness." Our research in this chapter addresses the role of the SAS Generator – devising new behavior streams as a response to non-routine problems. While SAS

Generator (Shallice, 1988) is a functional model to device solutions to non-routine problems, the presented mechanism here is a more concrete and detailed specification of the function of Shallice's SAS Generator; the specification is also unique in the sense that it tries to realize the function as per Baar's Global Workspace Theory in general and as the integrated operational cycle of IDA in particular.

6.5 Conclusion

To manage a nonroutine problem, first, the problem should be recognized, and then, a novel action plan (if there is one to find) should be produced as a solution to the problem. We presented a mechanism that addresses the later with the help of conscious mediation. The hypothesis is that "consciousness" provides an effective resource recruitment mechanism in the search process for a novel solution. This non-routine problem solver can be seen from the point of view of the human developmental process. We are born with basic primitive capabilities, which could correspond to the codelets in our mechanism or fine-grained level of competencies. We inherit some behaviors and develop other behaviors as experience in the environment progresses. The new behaviors are developed out of the established behaviors and the underling primitive capabilities. To start with, most problems are non-routine. Progressively, we accumulate routine problem solving capabilities via different ways: by instructions from other humans, by direct feedback from the interactions in the environment, by imitation, etc. In our mechanism, routine problem solving capabilities correspond to ready-made behavior streams (partial plans) in the procedural memory or template database (in the plan space). The behavior streams, the behaviors in them and the underlying behavior codelets are building blocks to generate solutions to non-routine problems. As experience grows, the non-routine problem becomes routine. If a routine task that uses a routine solution (behavior stream) is performed repetitively (or over learned), the routine task may become an automatic task; this is an automatization process, which we discussed in chapter 5. Automatization can be used as a foundation to chunk over learned sequences of actions into a larger unit, such as a behavior. The presented nonroutine problem solver is a type of adaptation or learning system that follows the Edlman's (1987) Neuronal Group Selection mechanism, which itself is inspired by Piaget's (1952) theory of cognitive development.

In general, planners produce a sequence of actions. In our case, the non-routine problem solving module, with a planner at the center, is used to produce a behavior stream template (or task network) that could be instantiated to perform actions in a certain sequence. The actual sequencing of the actions and conflict resolutions happen in the dynamics of the behavior net where instantiated streams reside. So, in IDA, the NRPS stream and the behavior net contribute in plan generation, and the behavior net does the plan execution. Both the plan generation and the plan execution are edited by "consciousness." Non-routine problem solving, as a deliberative process in IDA, is consciously mediated, and this enables IDA to interact with humans, using the service of its perception module. It is nearly impossible to preprogram the complete domain knowledge; so, for IDA and for any other agent system, an effective interaction with humans is important so that humans input their knowledge at contentious decisions.

We believe that integration of the AI planning system and the "consciousness" process, along with the human interaction capability in IDA, contribute towards the development of effective, non-routine problem solving systems.

7 Conclusion

This research contributes in three areas of the IDA model: (i) action selection mechanism, (ii) expectation, automatization and deautomatization mechanisms, and (iii) non-routine problem solving, and the corresponding details are presented in chapters 4, 5, and 6. In this chapter we will summarize what has been done, what has been postulated or hypothesized and what future work could be done in each of the three areas of research.

7.1 Main Contributions of Dissertation

This dissertation makes a number of contributions in cognitive computing by aspiring to model, design, and implement an integrated, cognitively plausible decision making mechanism action selection, expectation, automatization, deautomatization and non-routine problem solving. An overview of the contributions will be given below in that order.

7.1.1 Action Selection Mechanism

The design and implementation of IDA's action selection mechanism (chapter 4) puts into consideration the computer science and cognitive modeling constraints. The action selection mechanism realizes the goal context system postulated by global workspace theory. Besides, the design and implementation phases present a number of computer science issues such as communication and synchronization of distributed modules and reusability. The action selection module is fully implemented and tested in a simple domain. We propose to test the system in a relatively more complex domain.

7.1.1.1 Additions Over Maes' Behavior Net

IDA's action selection mechanism is based on Maes' new AI mechanism (1989), which is discussed in chapter three. Our mechanism has the following additions, which we argue are enhancements, over Maes' mechanism.

1. *Variables* – our mechanism deals with variables and instances of competencies that join the behavior net dynamics after instantiation of the corresponding schema or template of the competencies and the binding of their variables.

2. *Scope of search for selection* – In Maes' mechanism all the system competencies are part of behavior net action selection dynamics. In our mechanism, only the relevant goal context hierarchies are instantiated and become part of the action selection dynamics. We hypothesize that such "unconscious" mental search for actions is limited to the space of competencies that are found to be relevant. Search in limited space is computationally more efficient, which is likely being the case in evolving systems like humans. So, we conjecture that our approach is cognitively more plausible.

3. *Failure handling* – Maes' mechanism does not deal with failure situations; assumes every action produces the expected outcome. In the real world agents that operate in a dynamic environment will encounter failures on their actions. Our system deals with failures of behavioral actions using an expectation mechanism.

4. *Priority control* – The ability to prioritize competing goals (or tasks) is an important feature in decision making. The priorities may need to change from time to time. Our mechanism supports control of priorities for competing goal context hierarchies by using two new parameters (importance and discrimination-factor) that we introduced. The values of these parameters could be dynamically tuned to control priorities by mechanisms outside the behavior net. Components that could influence such decision processes in IDA's conceptual model are: (i) metacognition module and (ii) emotions and feelings module. Maes' mechanism does not support parametric control of such priorities. It could do so only by building causal orders among competence modules and such is not elegant and scalable at least for the following reasons: (i) the complete behavior net may need to be reprogrammed when ever a new behavior stream is integrated, (ii) priorities of behavior streams (tasks) could not be changed during run-time, and (iii) parametric control of task-level motivation is more plausible cognitive mechanism than explicit programming of priorities using causal links or rules.

5. *Planning and subgoaling* – Maes' mechanism could not support explicit AI planning (Maes, 1989) and subgoaling. Our behavior net mechanism, as a collection of goal structures, could support both. It accommodates "consciously" mediated actions; an example is the non-routine problem solving behavior stream we discussed in chapter 6.

7.1.1.2 A Goal Context Hierarchy System

In realizing the goal context system, major design and implementation issues have been resolved. The first issue to be considered is *representation.* We have shown that the action selection module is implemented as a distributed computing system with multiple levels of abstraction. In one abstract level (skybox), the action selection module is realized as a collection of behavior streams (goal context hierarchies), and each behavior steam is a partial action plan that handles a particular sub-problem. A behavior stream is represented by four types of objects: behavior nodes, goal nodes, propositions, and links. Behavior and goal nodes correspond to goal contexts and each has one or more classical variables, which are implemented as minimal frame data structures. Variables are bound during or after instantiation of the behavior streams. Each behavior is underlain by a coalition of processors: one or more behavior codelets and one or more expectation codelets. The action selection module is a motivated system. It has built-in motivations that are realized by drives. The different objects (behavior streams, behavior nodes, goal nodes, propositions, links, drives, behavior codelets, and expectation codelets) are implemented as independent threads. The action selection machinery has many more types of objects (see section 4.6), in which most are implemented as independent threads. The presented action selection mechanism has a novel representation and is a highly distributed system with true parallelism.

To adhere to global workspace theory, the behavior network mechanism, working as the goal context hierarchy system, and the "consciousness" mechanism should be coupled in such a way that a conscious event (via its broadcast of information content) influences which behavior (goal context) becomes active (dominant) in the immediate future. In turn, the execution of this dominant goal context should constrain what comes to "consciousness" next.

One of the major contributions of this research is to provide a mechanism that couples the consciousness and behavior network modules. Considering the 9-steps cognitive cycle (Table 2.1) of IDA, the action selection module realizes the last four steps (steps 6 to 9) of the cognitive cycle. As discussed in chapter 4, this research provides the design and implementation of the mechanisms of these steps. The steps show how conscious contents influence what behaviors (goal contexts) become active and control IDA's actions.

- A mechanism for the recruitment of relevant resources. This is implemented by behavior codelets with a priming mode and running as generator codelets. A generator behavior codelet, which is relevant to a "conscious" broadcast, dispatches an instance of itself (bound with the broadcast information). The instance behavior codelet jumps to the sideline. Thus, behavior codelets with a priming mode recognize their relevance to the situation that is defined in the "conscious" broadcast.

- A mechanism to set the relevant goal context hierarchy. Instances of a behavior codelets with a priming mode, working as recruited processors, instantiate the appropriate goal context hierarchies (behavior streams). The instantiation process includes: (i) the selection of templates or schema of the new behavior streams, (ii) the transformation of templates to running threads, (iii) the hooking of the new behavior streams into the behavior network dynamics (skybox), (iv) the binding by instance behavior codelets of the variables in the behavior streams they primed with the relevant information, and (v) the injection by instance behavior codelets of environmental activation to the directly linked behaviors of the behavior streams they primed.

- A mechanism to choose a dominant goal context (an appropriate behavior) for control of action. In chapter 4, we have shown how active behaviors (dominant goal contexts) control internal actions and external motor actions. The behavior network mechanism chooses a single behavior (goal context) to be active or dominant. The behavior network is a collection of instantiated behavior streams. Choosing the dominant goal context, based on a specific set of criteria, is done by the dynamics in the behavior network (skybox), which is influenced by: (i) motivating primary drives, (ii) current situation, (iii) the tuned parameter values, and (iv) relationships among behaviors and their history of activity patterns. The selection mechanism allows interleaving of goal context hierarchies – dominant goal context hierarchies could change in some order.

- The mechanism to take actions. While an instantiated behavior node competes in the dynamics of the behavior network (skybox), its underlying behavior codelets are instantiated in the sideline. An active or dominant behavior initializes its action by activating its underlying behavior and expectation codelets. Upon being activated, the behavior codelets and expectation codelets jump to the playing field. Once there, behavior codelets perform their actions, and expectation codelets feedback on the outcome of action. The behavior and expectation codelets use the variable binding information to perform their specialized tasks. The effects of the execution of behavior codelets are reflected by related updates in the working memory. The expectation codelets usually monitor these updates in the working memory.

Global workspace theory postulates that goal contexts constrain what comes to consciousness next. We have shown how this happens in Chapter 4. The action selection mechanism has its own working memory module; each of its memory items, which are similar to variables, is defined as a minimal frame data structure. Although it is a separate module, this working memory is part of the overall working memory of the agent. The effects of behavioral actions are reflected by relevant updates in the working memory in the same cognitive cycle or in the consecutive cognitive cycles. In one of the consecutive cycles, the outcome of external actions is updated in the working memory with the service of the cognitive steps 1 to 3: (1) perception of the effect of actions, (2) perception of effects of action to preconscious buffer, and (3) local associations. Once the outcome of an action is reflected in the working memory, a relevant attention codelet tries to bring the action outcome information to consciousness; i.e., an attention codelet competes for consciousness (cognitive step 4). If and when the attention codelet wins the competition, the action outcome information is broadcast (cognitive step 5). The conscious content recruits the appropriate resource and the cognitive cycle continues. The interaction of action selection and "consciousness" modules realizes a deliberative decision making system in which a "conscious" contents influences the unconscious goal contexts, and the unconscious goal contexts constrain the next "conscious" contents and this is what Baars (1988) call a C-U-C triad.

Our mechanism provides support for sophisticated action selection (decision making) types: automatic, voluntary, and consciously mediated. A behavior stream is a solution for a routine task. The behavior stream may perform the task automatically, i.e., without the service of "consciousness." Or it may perform the task with the mediation of "consciousness," in which the behavior stream needs the service of "consciousness" to provide pieces of information for the execution of some of its behaviors. Voluntary action (Franklin, 2000; Kondadadi & Franklin, 2001) is a "conscious" decision to take an action. Global workspace theory (Baars 1988) adopts James' (1890) ideomotor theory as a decision making mechanism for voluntary actions. Once a decision is made, a voluntary action is performed by the execution of the relevant behavior codelets.

One of the objectives as a computer science research is to separate the constant and the variable components of the action selection mechanism, so that we can isolate its domain independent features and build a reusable, general action selection framework. As discussed in section 4.6, the developed action selection framework simplifies the coding and representations of domain dependent knowledge pieces.

A lot of effort is put into the design and implementation of the action selection module as a reusable and general framework. One of the important reusability features is realized by the development a behavior network specification language called BNDL (Behavior Network Definition Language). BNDL simplifies the development of a behavior network. It could be considered as ontology for behavior networks. The BNDL specifications are compiled as text files with an XML format. The action selection mechanism uses its BNDL parser to convert the specification in the XML file to relevant machine executable objects.

The action selection module is implemented as a domain independent system and could even be used in a standalone agent framework to implement behavior based agents. We presented the steps required to design and implement a domain specific behavior network system.

We tested the action selection system with a relatively larger, real world simulated environment. The objective is to test our mechanism in which multiple drives and multiple behavior streams are instantiated at the same time and competing to each other. We intend to use a robot domain that operates in a warehouse floor. The warehouse stores different types of items. A khepera robot controller in a simulated environment is used to move unloaded items to shelving areas and to unshelve items move them to shipping areas. Each item type will have designated unloading, shelving and shipping areas. In this environment, the robot will have three drives: (i) go to charging area for recharging its battery, (ii) unload items and put them in the designated shelving area, and (iii) unshelve item move them to the designated shipping areas. Using the simulated robot environment, IDA's action selection mechanism was tested for performance and for control of priorities among instantiated goal context hierarchies. The results of our tests are:

1. GOMS analysis was done on the domain tasks and the performance of the robotic agent was close to the predicted performance value.

2. The effects of importance and discrimination-factor parameters on the control of priorities on competing goal context hierarchies were tested. These two and the other behavior net parameters were tuned to demonstrate successful control of priorities.

3. The quality of design and implementation of the action selection mechanism was evaluated by comparing the robot's performance with that of a human operator when both execute a task in the same level of knowledge constraint; the evaluation result was good.

4. The robotic agent, with relatively limited knowledge of the world, showed good performance compared to a human operator in performing a task. This is the case at least for the test domain we used.

7.1.2 Expectation, Automatization, and Deautomatization

The automatization and deautomatization functions and the corresponding mechanisms are discussed in chapter 5. The expectation mechanism is realized as an expectation codelet, which is discussed in section 4.3.3. Automatization develops with practice and its effect is to shift the consciously controlled processing of a procedural task to automatic processing. Automatization is an implicit learning process, which happens as a result of conscious experience. Automatized tasks can be performed without the service of conscious awareness unless there is a failure to observe an expected outcome in the execution of the task. The deautomatization process reinstates the service of conscious awareness when a failure to observe an expected outcome occurs. Reinstating conscious awareness helps to handle or edit the error situation in executing the task. We conjecture that the automatization/deautomatization mechanisms are related to attention and expectation of effects of actions.

The expectation, automatization and deautomatization functions are related to the two of the many functions of consciousness (Baars, 1997): (i) learning and adaptation and (ii) error-detection and editing (of action plans). Their corresponding mechanisms are important additions to the IDA model. All the mechanisms are fully designed and their implementations is well progressed.

As per our research objective of realizing the expectation, automatization and deautomatization mechanisms, we have addressed many issues including the following.

- We postulated that automatization is a multistep algorithmic process with some level of dependency on attention and expectation systems.

- The expectation mechanism was implemented and fully integrated into the action selection module. It is realized as expectation codelets with well defined functions. One or more expectation codelets underlie each behavior (goal context) and provide an anticipatory behavioral control mechanism.

- We developed a conceptual mechanism to have conscious access to expectations. Expectation codelets provide feedback on behavioral actions. An expectation codelet is also a type of attention codelet that could bring its feedback information to "consciousness." They always try to bring action errors to "consciousness."

- We proposed an operational condition for an automatization. That is, automatization happens only for actions (behaviors and the associated codelets) that do not involve attended change that cause variable binding. Or, automatization develops only for experiences on tasks performed in a situation attended to be invariant; i.e., the executed tasks are consistent and predictable.

- We developed the conceptual automatization mechanism so that "conscious" control will fade with experience. Experience is encoded as association strength among codelets. Automatization, as incidental learning process, is realized based on pandemonium theory (Jackson, 1987). Also, automatization, as a learning process, develops from conscious experience. In turn, consciousness access to tasks fades as a result of automatization. In our mechanism, fading of conscious access is realized by reducing the activation level (competitiveness) of relevant attention codelets as automatization develops. So, we conjecture that "conscious" experience facilitates learning; learning also shapes "conscious" experience.

- We developed a mechanism to detect action errors in automatized tasks. This is realized by expectation codelets. Automatization is related to expectations to action effects.

- We developed a deautomatization mechanism, which automatically breaks procedural skills into consciously accessible components when an error is observed in performing automatized tasks. This is realized by controlling a parameter to suspend the effect of automatization on relevant attention codelets. Suspension of the automatization effect decays away in a relatively short time. We hypothesize that deautomatization does not create new access points for "consciousness"; rather it temporarily boosts the "conscious" accessibility of existing attention points, that have been suppressed by the automatization effect.

- We showed how the automatization mechanism produces strong coordination among low-level processors or codelets in executing automatized procedural tasks. We showed that automatization qualitatively improves performance because automatized tasks require less processing in the behavior network and "consciousness" modules. That is, fewer cognitive steps are required to perform automatized tasks. We conjecture that automatization is an adaptation process to produce an optimally performing behavior or skill; the optimal performance arises from by passing relatively expensive mental processes like competition and access to limited resources.

- We have a mechanism to forget automatization or procedural skills. To do so, we use a decay mechanism with a variable decay rate at each instance codelet. The decay rate of a codelet in relation to the automatization is inversely proportional the amount of experience that builds the automatization. Experience is represented by the strongest association the codelet has with another codelet. An instance codelet that decays below a threshold will die, and as a result skills associated with that codelet are forgotten.

- We implemented the association mechanism and the different types of interactions. We integrated the automatization/deautomatization mechanisms in the attention, behavior, and expectation types of codelets. Namely, we implemented several channels for direct interactions ("unconscious" communication paths) among the different codelets. Again, the implementation involves issues of distributed computing since all codelets and most supporting modules run as independent threads.

- Our mechanism realizes the automatization and deautomatization cognitive functions as a self-organizing or an implicit-adaptation system – learning with out intention but as a result of "conscious" experience.

- We tested the automatization and deautomatization mechanisms in a simple domain and the results show that the implemented mechanism exhibits the required qualitative cognitive features - "conscious" access fades and performance improves with practice and deautomatization reinstate "conscious" access to task events when failure is encountered during execution of automatized procedures.

7.1.3 Non-Routine Problem Solving

In chapter 6, we discuss how the unavailability of ready made solutions (behavior streams) that respond to novel situations requires a capability to solve non-routine problems. Our research objective is to produce a detailed design of a non-routine problem solving mechanism to be integrated in the action selection and "consciousness" modules of the conceptual IDA model. We have shown how conceptual IDA would deliberate to generate clever, and possibly creative, solutions for non-routine problems. This calls for IDA to have a capability to detect impasses, to train attention on them, and to learn procedural methods based on related, known procedural knowledge pieces.

Towards the design of a mechanism for the non-routine problem solving, we have done the following:

- We defined the problem and provided preliminary ideas for a mechanism. We discussed how IDA could recognize non-routine problem situations and could be able to iteratively deliberate to solve them.

- We gave our design approach for detecting impasse or non-routine problem situations in the IDA model. In general, impasses or non-routine problems are detected at instances of failed expectations. Attention codelets detect impasses and bring them to "consciousness," which in turn recruit relevant resources that can assist in producing likely solutions. "Consciousness" increases the pull of relevant resources over multiple cycles.

- We presented our design approach to a mechanism for a non-routine problem solver (NRPS). That is, the NRPS uses available mid-level structures (behaviors) and primary repertoires (codelets) as building blocks to construct new goal structures (behavior streams) as solutions. Before it can produce a solution, the NRPS stream may need to be a dominant goal context hierarchy over multiple cognitive cycles. We postulate that problem solving is guided by special goal context hierarchies.

- We provided a detailed design for a non-routine problem solving mechanism for the IDA model; we fully specified the NRPS stream and discussed how the NRPS stream operates over multiple cognitive cycles to produce possible solutions.

- Intelligence relies on search — whether the search is for an intelligent agent's next external action or internal processing (thinking). We argue that "consciousness" is most likely to be an effective global searching tool to recruit relevant resources towards handling novel situation. We conjecture that NRPS behavior stream as a goal context hierarchy, along with a capability for human interaction, could be an effective non-routine problem solving system.

7.2 Future Work

A research work, as much as it adds our understanding to a given filed of study, it also provides new insights to see what is missing and allow us to formulate new questions. In this section we will discuss some of the improvements that could be made to what has been contributed in this dissertation. The automatization/deautomatization and non-routine problem solving mechanisms provide IDA a limited capability for adaptation. But, before IDA be called an adaptive agent, more learning capabilities in its decision making process that span over most (if not all) of the nine steps of its cognitive cycle, need to be incorporated. We briefly touch some of the cognitive computational research issues that are being addressed in the Learning IDA (LIDA) framework (D'Mello et al., 2006; Ramamurthy et al., 2006).

7.2.1 Action Selection Mechanism

Emotions and feelings modulate decision making and it is a line of research in the IDA research group. The objective is to develop a distributed motivational system using a mechanism for emotions and feelings. Future research could be to replace the role of drives as primary motivators to the behavior net module.

For each competency in the behavior net its activation level denotes its motivational strength. The parameters in the behavior net influence the activation/motivation dynamics. So, another future research work could be to map the role of each parameter in the dynamics and map them to appropriate emotional contents or feelings; this objective is to modulate decision making in the behavior net dynamics by automatic tuning of the parameters by the emotions and feelings.

For IDA to have an adaptive action selection system, learning capability need to be incorporated. Action selection related adaptation processes include: procedural, attentional, and expectation learnings. At present, the learning IDA (LIDA) is on going project for our research group, in which these and other learning types are being researched.

7.2.2 Expectation and Automatization

LIDA's conceptual model has a procedural learning mechanism (D'Mello et al., 2006). As an issue of an ongoing research (Negatu et al., 2006), necessary knowledge pieces, which underlie the realization new expectation codelets, should be acquired as part of the procedural learning. Such acquisition of knowledge allows the expectation mechanism to realize Hoffman's framework of anticipatory behavior control (Hoffman, 1993).

The automatization mechanism could be tested in relatively more complex domains, in which mutual interference exists among executing tasks and so as to study interference effects.

The automatization mechanism, with its automatic process of incremental learning to acquire skills and representational principles, could also be tested or adapted to accommodate the priming phenomena, which is understood to be a form of implicit learning (Becker et al., 1997).

7.2.3 Non-Routine Problem Solving

The immediate future work is to implement the NRPS (Non-Routine Problem Solving) behavior stream and the associated partial order planner as a special goal context hierarchy in the IDA model. Since the presented design specification is detailed and the required base components of the behavior net are in place, the implementation process could be completed relatively quickly.

7.3 As Mechanisms for Anticipation and Anticipatory Learning

According to Schaffer (1998) intelligence would be measured by the capacity for anticipation. While the role of anticipations on deliberation, memory, attention, behavior, and other facets of cognition has been well studied in cognitive psychology, neuropsychology, and ethology, the literature on explicit mechanisms to realize anticipations in artificial agents is considerably more sparse and scattered (Blank, Lewis, & Marshall, 2005; Butz, Sigaud, & Gerard, 2002; Kunde, 2001; Rosen, 1985; Schubotz & von Cramon, 2001). Over the last decade a variety of mechanisms that realize anticipations in artificial systems have been proposed (e.g., Drescher, 1991; Witkowski, 1997; Stolzmann, 1998; Blank et al, 2005). Studying the LIDA model in the anticipatory animat framework could allow us to have a different view of looking at the issue at hand - our endeavor to devise cognitively plausible integrated mechanisms for decision making and learning. LIDA integrates several cognitively inspired anticipation and anticipatory learning mechanisms (Negatu et al., 2006). Leaving the detailed study as a future work, below, we will briefly discuss some of these mechanisms.

7.3.1 Anticipation Mechanisms

Since anticipations have been acknowledged to be an influential component of the cognitive facilities of humans (and other animals), the need to model and integrate theories of anticipations in our artificial systems becomes vital. Butz, Sigaud, and Gerard (2002) provide several examples of such systems and have devised a useful nomenclature for the various anticipatory mechanisms that include *payoff, sensorial, state, and implicitly* anticipatory systems. The fundamental difference between implicit and the other three anticipatory systems is that in implicitly anticipatorial systems no explicit predictions about the future are made, even though the structure of the action selection component must contain certain anticipatory elements. Sensorial anticipation differs from payoff and state anticipatory mechanisms in that the predictions influence both early and later stages of sensory processing without directly having an impact on action selection. Finally, the main difference between payoff and state anticipatory mechanisms is that in payoff anticipatory systems anticipations play a role as payoff predictions only and explicit predictions of future states are not made. On the other hand state anticipatorial mechanisms make explicit predictions of future states during decision making processes.

7.3.1.1 Payoff Anticipatory Mechanisms

In a payoff anticipatory mechanism no explicit predictions of future states are made with the role of anticipations being restricted to some form of payoff, or utility, or reinforcement signal. In the L/IDA model the payoff for a behavior is calculated on the basis of predictive assessments by its *current activation* (i.e., relevance to the current goals or drives and environmental conditions) and its *base-level* activation (i.e., reliability in past situations)

LIDA's motivational system to influence goal-directed decision making is implemented on the basis of *drives*. Drives (sec. 4.2.1) are built-in or evolved (in humans or animals) primary and internal motivators. All actions are chosen in order to satisfy one or more drives, and a drive may be satisfied by different goal structures. A drive has an importance parameter (real value in [0,1]) that denotes its relative significance or priority compared to the other drives. Each drive has a preconditional proposition that represents a global goal. A drive spreads goal-directing motivational energy, which is weighted by the importance value, to behaviors that directly satisfy its global or deep goal. Such behaviors in turn spread activation backward to predecessor behaviors. Although external activation spreading includes situational motivation, in this discussion of anticipation, we will attend only to the action selection dynamics that are tuned to goal-end motivation. From this point of view, the current activation of a behavior at a given time represents the motivation level for its execution to satisfy sub-goals, which in turn contributes towards satisfying one or more global goals at some future time. In other words anticipating the predictive payoff in satisfying a goal influences the selection of the current action.

It should be noted that the use of a drive based motivation scheme in assessing the payoff in selecting a behavior may not clearly fit into one of the suggested distinctions of payoff vs. state anticipation. It has been suggested that such motivations and/or emotion systems, in influencing action decisions, indirectly predict states. Thus it could be argued that in reality these systems constitute a type of state anticipation.

The second factor that influences the payoff in selecting an action involves the use of the base-level activation of a scheme, which is a uninstantiated behavior in procedural memory. Behavior network scheme or procedural memory in LIDA (D'Mello et al., 2006a) is a modified and simplified form of Drescher's schema mechanism (1991), the scheme net. The scheme net is a directed graph whose nodes are (action) schemes and whose links represent the 'derived from' relation. Built-in primitive (empty) schemes directly controlling effectors are analogous to motor cell assemblies controlling muscle groups in humans. A scheme consists of an action, together with its context and its result. The context and results of the schemes are represented by perceptual symbols (Barsalou, 1999) for objects, categories, and relations in perceptual associative memory. The action of a scheme consists of one or more behavior codelets (discussed next) that execute the actions in parallel. The base-level activation is a measure of the scheme's overall reliability in the past, and is computed on the basis of the procedural learning mechanism described in the next section. It estimates the likelihood of the result of the scheme occurring after taking the action in its given context. When a scheme is deemed somewhat relevant to the current situation as a result of the attention mechanism, it is instantiated from the scheme template as a behavior into the action selection mechanism and allowed to compete for execution. This behavior shares the base-level activation of the scheme which, when aggregated with its current activation, produces a two-factor assessment of the anticipated payoff in selecting this behavior for execution. That is, goal-end motivation and past reliability produce anticipation value such that the satisfaction of deep goal(s) in the future and likelihood of success biases what action is to be executed during the current cycle.

7.3.1.2 State Anticipatory Mechanism

In the design of a state anticipatory mechanism we are concerned with explicit predictions of future states influencing current decision making. In LIDA, state anticipations come to play its non-routine problem solving (NRPS) process (chap. 6) – a deliberative process on par with the solution finding strategy called meshing (Glenberg, 1997). The NRPS process guides a controlled partial-order planner. While it shares similarities to dynamic planning systems it differs from earlier approaches such as the general problem solver (Newell, Shaw, & Simon, 1958) in that selective attention is used to target relevant solutions from procedural memory, thus pruning the search space on the basis of the current world model. Without going into the details, similar to any high-level planning system, the NRPS mechanism is a type of animat learning system that makes state anticipations, i.e., planning action decisions are biased towards selecting a plan operator that satisfies a required goal/sub-goal state.

7.3.1.3 Sensorial Anticipatory Mechanism

Rather than directly influence the selection of behaviors, sensorial anticipatory mechanisms influence sensorial processing (Butz, Sigaud, & Gerard, 2002). The LIDA system recognizes two forms of sensorial anticipation, the biasing of the senses similar to a preafferent signal (Freeman, 2001) and preparatory attention (LaBerge, 1995).

Nodes of the agent's perceptual associative memory, the slipnet (sec. 2.2), constitute the agent's perceptual symbols, representing individuals, categories and simple relations. Additionally, schemes in the agent's procedural memory represent uninstantiated actions and action sequences. The context and results of the schemes are represented by the same nodes for objects, categories, and relations in perceptual associative memory. A behavior in the behavior network is an instantiated scheme, thereby sharing its context (as preconditions) and results (as postconditions). In LIDA, once a behavior is selected in the behavior net, the nodes of the slipnet that compose the postconditions of the behavior have their activations increased, thus biasing them towards selection in the next cycle.

Preparatory attention in LIDA is also implemented on the basis of the currently selected behavior. Each behavior is equipped with one or more expectation codelets, a special type of attention codelet that attempts to bring the results of selected action to attention. Once a behavior is selected for execution, its expectation codelets attempt to bring the results of the behavior to attention, thereby biasing selective attention. In this manner the LIDA system incorporates a second form of action driven sensorial anticipation.

7.3.2 Anticipatory Learning

Here, we explore an automatization mechanism to learn low-level implicit anticipations, and a procedural learning mechanism to learn the context and results of existing actions, which in turn, are used to construct a variety of anticipatory links

The automatization mechanism implicitly causes a controlled task execution process to transition into a highly coordinated skill thus improving performance and reserving attention, a limited resource, for more novel tasks (chap. 5). It is a type of implicit anticipatory learning mechanism since the encoding of the experiences of performing tasks is integrated in, and arises from, the payoff anticipatory process of LIDA's action selection dynamics. That is, on the basis of the automatization mechanism implicitly anticipatory links among the low-level processors (codelets) are learnt (buying optimality in task execution) as a result of experiencing anticipatory (payoff) decision making at the high level constructs (behaviors).

Anticipatory learning also takes place during the creation of new schemes. Adaptive agents are usually equipped with a capability to generate exploratory actions. Such action generations at the beginning are based on random (trial and error) and with a motivation of a curiosity drive. In LIDA, for creation or learning of a new procedure to proceed, the generation of exploratory behavior means that the behavior network must first select the instantiation of an empty scheme for execution. Before executing its action, the instantiated scheme spawns a new expectation codelet. After the action is executed, this newly created expectation codelet focuses on changes in the environment that result from the action being executed, and attempts to bring this information to attention. If successful, a new scheme is created, if needed. If one already exists, it is appropriately reinforced. Perceptual information selected by attention just before and after the action was executed form the context and result of the new scheme respectively. The scheme is provided with some base-level activation, and it is connected to its parent empty scheme with a link. More details on this mechanism can be found in (D'Mello et al., 2006). The creation of a new scheme leads to a number of new anticipatory links being formed. The result of the scheme can be used to learn new expectation codelets to monitor future execution. These expectation codelets can be used to assess the reliability of this scheme thus influencing payoff anticipations. They also serve as sensorial anticipations by biasing perceptual associative memory and selective attention.

7.4 Summary

In this concluding chapter we have described: (i) the main contribution of this dissertation towards the development of a cognitively inspired decision making mechanisms – action selection (Negatu & Franklin, 2002), expectation, automatization/deautomatization (Negatu, McCauley, & Franklin, in review), and non-routine problem solving (McCauley, Negatu, & Franklin, in preparation); (ii) future work that could be continued to forward our main research issues; and (iii) the anticipation and anticipatory mechanisms that are integrated the L/IDA agent architecture (Negatu, D'Mello, & Franklin, in review). Although not discussed in this document, as part of our software agent research, we have made investigation on learning concepts and mechanisms (Ramamurthy, Negatu, & Franklin, 1998, 2001; Negatu & Franklin, 1999; D'Mello et al., 2006). Our collaborative endeavor in furthering the modeling and implementation phases of our computational agent system has been rewarding. It allows us to have a better understanding of the challenges and to ask better research questions in our core field of study - computer science; in various degrees, we also become inquisitors and conversants in other fields of studies including cognitive psychology, cognitive neuroscience and ethology.

Bibliography

Agre, P. & Chapman, D. (1987). Pengi: An Implementation of a Theory of Activity. *Proceedings of Sixth National Conference on AI*, AAAI-87. Los Altos, CA: Morgan Kaufmann.

Albus, JS (1981). *Brains, Behaviours & Robotic*s. BYTE Publications.

Allen, J. J. (1995). *Natural Language Understanding*. Redwood City, CA: Benjamin/Cummings Benjamin Cummings.

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1992). Automaticity and the ACT theory. *American Journal of Psychology*, 105, 165-180.

Anderson, J. R. (1993). Problem solving and learning. *American Psychologist*, 48, 35-44.

Anderson, J. R. (1995). Developing Expertise. In J. R. Anderson (Ed.), Cognitive psychology and its implications (rth ed.) (pp. 274-304). New York: W.H. Freeman.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist*, 51, 355-365.

Anderson, J., & Lebiere, C. (1998) *Atomic Componenets of Thought*. Lawrence Erlbaum.

Anderson, J. R., Matessa, M., & Lebiere, C. (1997). ACT-R: A theory of higher level cognition and its relation to visual attention. *Human Computer Interaction*, 12(4), 439-462.

Baars, B.J. (1988). *A Cognitive Theory of Consciousness*. Cambridge: Cambridge University Press.

Baars, B.J. (1997). *In the Theory of Consciousness*. Oxford: Oxford University Press.

Baars, B.J. (2002). The conscious access hypothesis: origins and recent evidence. *TRENDS in Cognitive Sciences,* Vol. 6 No. 1, (PP. 47-52). Jan. 2002.

Baars, B.J. (2000). Treating consciousness as a variable: The fading taboo. In Bernard J. Baars, William P. Banks and James Newman, editors, *Essential Sources in the Scientific Study of Consciousness,* Chapter 1, The MIT Press, Bradford Books.

Baars, B. J., & Franklin, S. (2003). How conscious experience and working memory interact. *Trends in Cognitive Science, 7*, 166-172.

Baddeley, A., Conway, M., and Aggleton, J. (2001) *Episodic Memory*, Oxford: Oxford Univeristy Press.

Bargh, J.A. 1992. The Ecology of Automaticity: Towards establishing the conditions needed to produce automatic processing effect. *American Journal of Psychology*, 105, 181-199.

Bargh, J.A. & Chartrand, T. L. (1999). The Unbearable Automaticity of Being. *American Psychologist*, Vol. 54, No. 7, 462-479.

Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences* 22:577-609.

Beard, D. V., Dana K. Smith, & Kevin M. Denelsbeck. (1996). QGOMS: a direct manipulation tool for simple GOMS models. In conference *Companion on Human factors in computing systems*: common ground, Vancouver, Canada.

Becker, S., Moscovitch, M., & Joordens, S. (1997). Long-term semantic priming: a computational account and empirical evidence. *Journal of Experimental Psychlogy*: Learning, Memory and Cognition, 23, 1059-1082.

Beer, R. (1990). *Intelligence as Adaptive Behavior*. Academic press, NY.

Beer, R., & Chiel, H.1991. The neural basis of behavioral choice in an artificial insect. In Meyer, J.-A. and Wilson, S., editors, *From Animals to Animats: The First International Conference on Simulation of Adaptive Behavior*, pp. 247-254. The MIT Press, Cambridge, MA.

Block, N. (2002). “Some Concepts of Consciousness” In *Philosophy of Mind: Classical and Contemporary Readings*, David Chalmers (Ed.) Oxford University Press, 2002.

Blumberg, B.1994. Action-selection in hamsterdam: Lessons from ethology. In Cliff, D., Husbands, P., Meyer, J.-A., and S, W.,(Eds), From Animals to Animats: The Third International Conference on *Simulation of Adaptive Behavior,* pp. 108-117. The MIT Press, Cambridge, MA.

Bogner, M. (1999). Realizing "consciousness" in software agents. PhD Dissertation. University of Memphis.

Bogner, M., U. Ramamurthy, and S. Franklin. (1999). “Consciousness" and Conceptual Learning in a Socially Situated Agent. In *Human Cognition and Social Agent Technology*, Advances in Consciousness Research Series, 19. (Ed. K). Dautenhahn. Amsterdam: John Benjamins.

Bovair, S., Kieras, D.E., & Polson, P.G. (1990). The acquisition and performance of text editing skill: A cognitive complexity analysis. *Human-Computer Interaction*, **5**, 1-48.

Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pages 12-23.

Blank, Douglas S., Lewis, Joshua M., and Marshall, James B. (2005). The Multiple Roles of Anticipation in Developmental Robotics. AAAI Fall Symposium Workshop Notes, *From Reactive to Anticipatory Cognitive Embodied Systems*. AAAI Press

Brooks, R. A. (1990) "A Robot That Walks: Emergent Behaviors from a Carefully Evolved Network." In P. H. Winston, ed., *Artificial Intelligence at MIT*, Vol. 2. Cambridge, Mass.: MIT Press.

Butler, K. A., Bennett, J., Polson, P., and Karat, J. (1989). Report on the workshop on analytical models: Predicting the complexity of human-computer interaction. *SIGCHI Bulletin*, 20(4), pp. 63-79.

Butz, M. V., Sigaud, O., & Gerard, P. (2002). Internal models and anticipations in adaptive learning systems. In *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems*, 1–23.

Cañamero, D. (1997). Modeling Motivations and Emotions as a Basis for Intelligent Behavior. In *Proceedings of the First International Symposium on Autonomous Agents, AA'97*, Marina del Rey, CA, February 5-8, The ACM Press.

Chalmers, D. J. (1996). *The Conscious Mind.* Oxford: Oxford University Press.

Chartrand, Tanya L. & Bargh, John. (1996). "Automatic Activation of Impression Formation and Memorization Goals: Nonconscious Goal Priming Reproduces Effects of Explicit Task Instructions." *JNL of Personality and Social Psychology* (1996). Vol. 71. No. 3. 464-478.

Conway, M. A. (2001) Sensory-perceptual episodic memory and its context: autobiographical memory. In *Episodic Memory*, ed. A. Baddeley, M. Conway, and J. Aggleton. Oxford: Oxford University Press.

Damasio, A. R. (1994). *Descartes' Error.* New York: Gosset; Putnam Press.

Decugis, V. & Ferber, J. (1998). Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. *Proceedings of the second international conference on autonomous agents.* Minneapolis, MN USA.

De Keyser, R. (2001). "Automaticity and Automatization" In: Robinson, Peter (Ed.): Cognition and Second Language Instruction. New York. 125-151.

D'Mello, S. K., Franklin, S., Ramamurthy, U., & Baars, B. J. (2006). A Cognitive Science Based Machine Learning Architecture. AAAI Spring Symposia Technical Series, Stanford CA, USA. Technical Report SS-06-02 (pp. 40-45). AAAI Pres.

D'Mello, S. K., Ramamurthy, U., Negatu, A. S., & Franklin, S. (2006). A Procedural Learning Mechanism for Novel Skill Acquisition. Workshop on Motor Development, part of AISB'06: Adaptation in Artificial and Biological Systems, University of Bristol, Bristol, England April 2006.

Doignon, J.-P., & Falmagne, J.-C. (1985). Spaces for the assessment of knowledge. *International Journal of Man-Machine Studies, 23*, 175-196.

Dorer, Klaus. (1999). Behavior Networks for Continuous Domains using Situation-Dependent Motivations. In *International Joint Conference on Articial Intelligence* (IJCAI'99), pages 1233-1238.

Drescher, G. (1991). Made Up Minds: A Constructivist Approach to Artificial Intelligence, Cambridge, MA: MIT Press.

Ebbinghaus, H. (1885). Über das Gedachtnis: Untersuchungen zur Experimentellen Psychologie. Leipzig, Germany: Duncker & Humblot.

Edelman, G. M. (1987). *Neural Darwinism: the theory of neuronal group selection.* New York: Basic Books.

Edelman, G. M. & Tononi, G. (2000). A Universe of Consciousness. New York: Basic Books.

Engel, A.K., Fries, P., Konig, P., Precht, M., & Singer, W. (1999). Temporal Binding, binocular rivalry, and consciousness. *Consciousness and Cognition*, 8, 128-151.

Ericsson, K. A., & Kintsch, W. (1995). Long-term working memory. *Psychological Review* 102:21–245.

Erol, K.; Hendler, J.; & Nau, D.S. (1994). UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proc.* AIPS. Morgan Kaufman.

Flavell, J. H. (1976). Metacognitive aspects of problem solving. In L. B. Resnick (Ed.), *The nature of intelligence.* Hillsdale, NJ: Erlbaum.

Franklin, S. (1995). *Artificial Minds.* Cambridge MA: MIT Press.

Franklin, S. (1997). Autonomous Agents as Embodied AI. *Cybernetics and Systems* 28:499–520.

Franklin, S. (2000). Deliberation and Voluntary Action in 'Conscious' Software Agents. *Neural Network World* 10:505-521.

Franklin, S. (2001). Automating Human Information Agents. In *Practical Applications of Intelligent Agents*, ed. Z. Chen, and L. C. Jain. Berlin: Springer-Verlag.

Franklin, S. (2001b). An Agent Architecture Potentially Capable of Robust Autonomy. *AAAI Spring Symposium on Robust Autonomy*; American Association for Artificial Intelligence; Stanford, CA; March.

Franklin, S., & L. McCaulley (2002). Feelings and Emotions as Motivators and Learning Facilitators. (*Architectures for Modeling Emotions.* AAAI Spring Symposia Technical Series Technical Reports; SS-04-02)

Franklin, S. 2003. IDA: A Conscious Artifact? *Journal of Consciousness Studies* 10:47-66. Holland, O.; 2003; ed.; Machine Consciousness; Journal of Consciousness Studies

Franklin, S., & Graesser, A.C. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III.* Berlin: Springer Verlag.

Franklin, S., & A. Graesser. (1999). A Software Agent Model of Consciousness. *Consciousness and Cognition* 8:285-305.

Franklin, S., & A. Graesser. (2001). Modeling Cognition with Software Agents. In *CogSci2001: Proceedings of the 23rd Annual Conference of the Cognitive Science Society*, ed. J. D. Moore, and K. Stenning. Mahwah, NJ: Lawrence Erlbaum Associates; August 1-4, 2001.

Franklin, S. Kelemen, A. & McCauley, L.; (1998). IDA: A Cognitive Agent Architecture. *IEEE Conference on Systems, Man and Cybernetics.*

Franklin, S., Baars, B.J., Ramamurthy, U., & Ventura, M. (2005). The Role of Consciousness in Memory" in *Brains, Minds and Media*, Vol.1, 2005 (bmm150) (urn:nbn:de:0009-3-1505).

Freeman, W.J. (1995). Societies of Brains. *A Study in the Neuroscience of Love and Hate.* Hillsdale NJ: Lawrence Erlbaum.

Frijda, N. H. (1986). *The Emotions*. Cambridge: Cambridge University Press.

Glenberg, A. (1997). What memory is for. *Behavioral and Brain Sciences, 20*, 1-19.

Glenberg, A., & Robertson, D. A. (2000). Symbol grounding and meaning: A comparison of high-dimensional and embodied theories of meaning. *Journal of Memory and Language, 43*, 379-401.

Gunzelmann, G., & Anderson, J. R. (2003). Problem solving: Increased planning with practice. Cognitive Systems Research, 4, 57-76.

Hacker, D.J. (1998), 'Metacognitive: Definitions and Empirical Foundations' in Hacker, D.J., Dunlosky, J., & Graesser, A.C. (Eds.) *Metacognition in educational theory and practice.* Mahwah, NJ: Erlbaum.

Hoffman, J. (1993). Vorhersage und erkenntnis [Anticipation and Cognition]. Hogrefe.

Hofstadter, D. R. (1995). *Fluid Concepts and Creative Analogies.* : Basic Books.

Hofstadter, R. D., & Mitchell, M. (1994). The Copycat Project: A model of mental fluidity and analogy-making. In *Advances in connectionist and neural computation theory, Vol. 2: Analogical connections, eds. K. J. Holyoak & J. A. Barnden.* Norwood N.J.: Ablex.

Holland, J. H. (1986). A Mathematical Framework for Studying Learning in Classifier Systems. In *Evolution, Games and Learning: Models for Adaption in Machine and Nature*, vol. al, Amsterdam, ed. D. Farmer. : North-Holland.

Jackson, J. V. (1987). Idea for a Mind. Siggart Newsletter, 181:23–26.

James, W. (1890). *The Principles of Psychology.* Cambridge, MA: Harvard University Press.

John, B. E. & Kieras, D. E. (1994) The GOMS family of analysis techniques: Tools for design and evaluation. Carnegie Mellon University School of Computer Science Technical Report No. CMU-CS-94-181. Also appears as the Human-Computer Interaction Institute Technical Report No. CMU-HCII-94-106.

John, B. E., & Kieras, D. E. (1996a). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction*, 3, 287-319.

John, B. E., & Kieras, D. E. (1996b). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, 3, 320-351.

Johnson, T. R. (1997). Control in Act-R and Soar. In M. Shafto & P. Langley (Eds.), *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society* (pp. 343-348): Hillsdale, NJ: Lawrence Erlbaum Associates.

Jost, A. (1897). Die Assoziationsfestigkeit in ihrer Abha¨ngigkeit von der Verteilung der Wiederholungen [The strength of associations in their dependence on the distribution of repetitions]. *Zeitschrift fur Psychologie und Physiologie der Sinnesorgane, 16,* 436–472.

Kahneman, D., & Chajczyk, D. 1983. Test of automaticity of reading: Dilution of the stroop effect by color-irrelevant stimuli. *Journal of Experimental Psychology: Human Perception and Performance*, 9, 947-509.

Kanerva, P. (1988). Sparse Distributed Memory. Cambridge MA: The MIT Press.

Kay, L.M. & Freeman, W.J. (1998). Bidirectional processing in the olfactory-limbic axis during olfactory behavior. *Behavioral Neuroscience*, 112(3): 541-553

Kelemen, A., Y. Liang, R. Kozma, & S. Franklin. (2002). Optimizing Intelligent Agent's Constraint Satisfaction with Neural Networks. In *Innovations in Intelligent Systems*, ed. A. Abraham, and B. Nath. Heidelberg, Germany: Springer-Verlag,

Kieras, D. E. (1997). A Guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, and P. Prabhu (Eds.), Handbook of human-computer interaction. (Second Edition). Amsterdam: North-Holland. 733-766.

Kieras, D. E., Meyer, D. E., Mueller, S., & Seymour, T. (1999). Insights into working memory from the perspective of the EPIC architecture for modeling skilled perceptual-motor performance. In P. Shah & A. Miyake (Eds.) *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, Cambridge: Cambridge University Press.

Kieras, D. E., Meyer, D. E., Mueller, S., & Seymour, T. (1998). An EPIC Computational Model of Working Memory. *Poster presented at the 39th Annual meeting of the Psychonomics Society.*

Kieras, D. E. (2003). GOMS Models for Task Analysis. In D. Diaper, Neville A. Stanton, editors. *The Handbook of Task Analysis for Human-Computer Interaction.* Lawrence Erlbaum Associates 83-116.

Kintsch, W. 1998. Comprehension. Cambridge: Cambridge University Press.

Kondadadi, R. & Franklin, S. (2001). A Framework of Deliberative Decision Making in "Conscious" software Agents. In *Proceedings Of Sixth International Symposium on Artificial Life and Robotics* (AROB-01).

Kokiniv, B. (1994a). A hybrid model of reasoning by analogy. In K. Holyoak and J. Barnden (Ed). *Advances in connectionist and neutral computation theory. Vol. 2: Analogical connections.* Norwood, NJ: Ablex.

Kokiniv, B. (1994b). The DUAL cognitive architecture: A hybrid multi-agent approach. *Proceedings of the Eleventh European Conference on AI.* Wiley.

Kunde, W. (2001). Response–effect compatibility in manual choice reaction tasks. *Journal of Experimental Psychology: Human Perception & Performance*, 27, 387-394.

Laird, E. J., Newell, A. & Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33:1–64.

Logan, G. D. (1980). Attention and Automaticity in stroop and priming tasks: Theory and data. *Cognitive Psychology*, 12, 525-553.

Logan, G. D. (1985). Skill and automaticity: Relations, implications and future direction. *Canadian Journal of Psycology, 39,* 367-386.

Logan, G. & Cowan, W. (1984). On the ability to inhibit thought, and action: A theory of an act of control. *Psychology Review, 91,* 295-327.

Logan, G.D. (1998). Toward an instance theory of automatization. *Psychological Review*, 95, 583-598.

Ludlow, A.1980. The evolution and simulation of a decision maker. In Toates, F. and Halliday, T., (Eds.), *Analysis of Motivational Processes.* Academic Press, London.

Maes, P. (1989). How to do the right thing. *Connection Science, 1*, 291-323.

Matessa, M. & Anderson, J. R. (2000). An ACT-R model of adaptive communication. In *Proceedings of the Third International Conference on Cognitive Modeling*, pp. 210-217, University of Groningen, Netherlands.

Maturana, R. H., & F. J. Varela. (1980). Autopoiesis and Cognition: The Realization of the Living, Dordrecht. Netherlands: Reidel.

Maturana, H. R. (1975). The Organization of the Living: A Theory of the Living Organization. *International Journal of Man-Machine Studies* 7:313–332.

McCauley, T. L., & Franklin, S. (1998). An Architecture for Emotion. *AAAI Fall Symposium Emotional and Intelligent: The Tangled Knot of Cognition"*; AAAI; Orlando, FL.

McCauley, L., Franklin, S. & Bogner, M. (2000). An Emotion-Based "Conscious" Software Agent Architecture. In *Affective Interactions, Lecture Notes on Artificial Intelligence ed.*, vol. 1814, ed. A. Paiva. Berlin: Springer.

McCauley, L., A. S. Negatu, & S. Franklin. (in preparation). An Agent Architecture for Non-Routine Problem Solving.

Meyer, D. E., & Kieras, D. E. (1999). Precis to a Practical Unified Theory of Cognition and Action: Some Lessons from EPIC Computational Models of Human Multiple-Task Performance. In D. Gopher & A. Koriat (Eds.) *Attention and Performance XVII. Cognitive Regulation of Performance: Interaction of Theory and Application.* (pp. 17-88). Cambridge, MA: MIT Press, 1999.

Minsky, M. (1985). *The Society of Mind.* New York, NY: Simon and Schuster.

Mitchell, M. (1993). *Analogy-Making as Perception.* Cambridge, MA: MIT Press.

Negatu, A. S., & Franklin, S. (1999). Behavioral Learning for Adaptive software Agent. *Intelligent Systems, Proceeding of the ISCA 8th International Conference,* pp. 91 – 95, Denver, Colorado.

Negatu, A. S., & S. Franklin. (2002). An action selection mechanism for 'conscious' software agents. *Cognitive Science Quarterly* 2:363-386.

Negatu, A. S., T. L. McCauley, & S. Franklin. (in review). Automatization for Software Agents.

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

Nii, H. P. (1986). "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architecture", *AI Maagazine 7 (2).*

Oatley, K., & Jenkins, J. M. (1996). *Understanding Emotions.* Cambridge, MA: Blackwell Publishers Ltd.

Olson, J. R., & Olson, G. M. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. Human-Computer Interaction, 5, 221-265.

Ornstein, R. (1986). *Multimind.* Boston: Houghton Mifflin.

Piajet, J. (1952). *The origins of intelligence in children.* Intl. Univ. Press.

Picard, R. W. (1997). *Affective Computing.* Cambridge, MA: The MIT Press.

PlihalW, Born J. 1997. Effects of early and late nocturnal sleep on declarative and procedural memory. *J. Cogn. Neurosci.* 9:534–47

Povinelli, D. (1994) What chimpanzees know about the mind. In Behavioral Diversities in chimpanzees. Harvard University Press.

Pylyshyn, Z.W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science.* MIT Press

Ramamurthy, U., S. Franklin, & A. S. Negatu. (1998). Learning Concepts in Software Agents. In *From animals to animats 5: Proceedings of The Fifth International Conference on Simulation of Adaptive Behavior*, ed. R. Pfeifer, B. Blumberg, J.-A. Meyer, and S. W. Wilson. Cambridge,Mass: MIT Press.

Ramamurthy, U., Baars, B., D'Mello, S. K., & Franklin, S. (2006). LIDA: A Working Model of Cognition. *The 7th International Conference on Cognitive Modeling*, Trieste, Italy, April 2006.

Reber, A. S. (1989) More thoughts on consciousness. Reply to Brody and to Lewicki and Hill. *Journal of Experimnetal Psychol.* Gen. 118, 244-244.

Rhodes, B. (1995). Pronomes in Behavior Nets. *Technical Report # 95-01*. MIT Media Lab, MIT, Mass.

Rhodes, B. (1996). *PHISH-Nets: Planning Heuristically in Situated Hybrid Networks*. MS thesis. School of Architecture and Planning, Massachusetts Institute of Technology, September 1996.

Rosenblatt, J., & Payton, D. W. (1989). A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control. *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, Washington DC, June 1989, Vol. 2, pp. 317-324.

Rosen, R. (1985). *Anticipatory Systems*. Pergamon Press.

Russell, s & Norvig, P. (2003). *Artifficial Intelligence A Modern Approach.* 2nd Ed. Pearson Education, Inc.

Sacerdoti, E.D. (1977). *A Structure for Plans and Behavior*, Elsevier-North Holland.

Schaffer, S. (1998). Babbage's Intelligence: Calculating Engines and the Factory System, hosted at http://cci.wmin.ac.uk/schaffer/schaffer01.html (1998).

Schneider, W. & Fisk, A.D. 1983. Attentional theory and mechanisms for skilled performance. In R.A. Magill (Ed.), *Memory and control of action* (pp. 119-143). New York: North-Holland.

Schneider, W. & Shiffrin, R. (1977). Controlled and automatic human information processing: I. Perceptual learning, automatic attending, and a general theory. *Psychlogical Review 84*, 127-190.

Schubotz, R.I., von Cramon, D.Y. (2001). *Functional organization of the lateral premotor cortex.* fMRI reveals different regions activated by anticipation of object properties, location and speed. Cognitive Brain Research 11 97-112

Shastri, L. (2202), Episodic memory and cortico-hipocampalinteractions. *Trends Cogn. Science* 6, 162-168.

Shiffrin, R. & Schneider, W. (1977). Controlled and automatic human information processing: II. Detection, search, and attention. *Psychlogical Review 84*, 1-66.

Sloman, A. (1987). Motives Mechanisms Emotions. *Cognition and Emotion* 1:217–234.

Sloman, A. (1999). What Sort of Architecture is Required for a Human-like Agent? *In Foundations of Rational Agency*, ed. M. Wooldridge, and A. Rao. Dordrecht, Netherlands: Kluwer Academic Publishers.

Smith, D.C., Cypher, A., & Spohrer, J. (1994). KidSim: Programming agents without a programming language. *Communication of the ACM.* Vol. 7, Issue 7, pp. 54-67.

Song, H., Franklin, S., & Negatu, A. S. (1996). SUMPY: A Fuzzy Software Agent. In Proceedings of *International Conference on Intelligence Systems*, pp 124-129, Nevada.

Song, H., & Franklin, S. (2000). A Behavior Instantiation Agent Architecture. *Connection Science* 12:21-44.

Stolzmann, W. (1998). *Anticipatory Classifier Systems.* in Genetic Programming. University of Wisconsin, Madison, Wisconsin: Morgan Kaufmann

Sun, R., & S. Franklin. 2004. Computational Models of Consciousness: A Taxonomy and some Examples. In *Cambridge Handbook of Consciousness*, ed. P. D. Zelazo, and M. Moscovitch. New York: Cambridge University Press.

Tate, A. (1990). Generating Project Networks. In Alen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 291-296.

Tate, A., Drabble, B., & Kirby, R. B. (1994). O-Plan2: an open architecture for command, planning, and control. In Fox, M, and Zweben, M., editors, *Intelliegent Scheduling.* Morgan Kaufman, San Francisco, CA. 213-239.

Tinbergen, N (1950). The hierarchical organization of mechanisms underlying instinctive behaviour. *Experimental Biology,* 4, 305–312.

Tulving, E., & Madigan, S.A. (1970). Memory and verbal learning. Annual Review of Psychology, 21, 437–484.

Tyrrell, T. (1992). Defining the action selection problem. Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society.

Tyrell, T. (1993). Computational Mechanisms for Action Selection. *PhD Thesis*. University of Edinburg, UK.

Tzelgov, J, (1997a). Automatic but conscious: That is how we act most of the time. In R.S. Wyer, Jr. (Ed), *Advances in social cognition (Vol. X, pp. 217-230).* Mahwah, NJ: Erlbaum.

Tzelgov, J, (1997b). Specifying the relations between automaticity and consciousness: A theoretical note. *Consciousness and cognition, 6,* 441-451.

Underwood BJ. 1957. Interference and forgetting. *Psychol. Rev.* 64:49–60

Underwood, B. J., & Postman, L. (1960). Extraexperimental sources of interference in forgetting. *Psychological Review, 67,* 73–95.

Velásquez, J. (1997). Modeling Emotions and Other Motivations in Synthetic Agents. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press.

Wegner, Daniel M. & Bargh, John A. "Control and Automaticity in Social Life", in *Handbook of Social Psychology* (4/e). D. Gilbert, S. Fiske, & G. Lindzey (eds). McGraw-Hill:1998.

Whitetaker, H, (1983). Towards a brain model of automatization: A short essay. In R. A. Magill (Ed.), Memory and Control of Action (pp. 194-214). Yew York: North-Holland.

Wilkins, D. E. (1990a). Domain-independent Planning: Representation and Plan generation. In Alen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning*. Morgan Kaufman. 319-335.

Wilkins, D.E. (1990b). Can AI planners solve practical problem? *Computational Intelligence* 6(4): 232-246.

Wilkins, D.E., Myers, K.L., Lowrance, J.D., & Wesley, L.P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI.* &(1): 197-227.

Wilkins, E. D. (2001). A Call for Knowledge-based Planning. *AI Magazine*, Spring 2001, Vol. 22, No. 1.

Witkowski, C. M., (2002). Anticipatory Learning: The Animat as Discovery Engine. In Brutz, Sigaud, and Gerard (Eds.), Adaptive Behavior in Anticipatory Learning System (ABiALS'02), Edinburgh, Scotland.

Wixted, J. T. (2004a). On common ground: Jost's (1897) law of forgetting and Ribot's (1881) law of retrograde amnesia. *Psychological Review, 111*, 864-879.

Wixted, J. T. (2004b). The psychology and neuroscience of forgetting. *Annual Review of Psychology, 55*, 235-269.

Woodworth, R. S. (1938). *Experimental psychology.* Oxford, England: Holt.

Zhang, Z., D. Dasgupta, & S. Franklin. (1998a). Metacognition in Software Agents using Classifier Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, Wisconsin: MIT Press.

Zhang, Z., S. Franklin, B. Olde, Y. Wan, & A. Graesser. (1998b). Natural Language Sensing for Autonomous Agents. In *Proceedings of IEEE International Joint Symposia on Intellgence Systems 98.*

A Behavior Network Definition Language (BNDL) Grammar

```
<?xml version='1.0' encoding='us-ascii'?>

<!-- File bndl_grammar_r_v2.dtd -->
<!-- Behavior Net Description Language (BNDL) grammer - Document Type
Definition -->
<!-- Version 1.0 -->
<!-- Developed by: Aregahegn S. Negatu -->

<!ELEMENT behaviornet (stream+, drive+, bcodelet*, dm-type*, dm-
instance*, general-info*)>
<!ATTLIST behaviornet
            element-name    NMTOKEN    #REQUIRED
            theta     NMTOKEN    #REQUIRED
            gamma     NMTOKEN    #REQUIRED
            pi        NMTOKEN    #REQUIRED
            delta     NMTOKEN    #REQUIRED
            phi       NMTOKEN    #REQUIRED
            decay-rate NMTOKEN   #IMPLIED
            >

<!ELEMENT stream (behavior+, goal+, primingrule*)>
<!ATTLIST stream
            element-name NMTOKEN #REQUIRED
            >

<!ELEMENT behavior (alias*, precondition+, bcodelet*, expcodelet*,
attncodelet*, (addition | deletion)+, variable*, motor-variable*,
perception-variable*) >
<!ATTLIST behavior
            element-name    NMTOKEN    #REQUIRED
            mode            (true | false) #REQUIRED
            type            (action | inference) #REQUIRED
            codelet-names  IDREFS  #IMPLIED
            >

<!ELEMENT goal (alias*, precondition+, attncodelet*, (addition |
deletion)+, variable*, motor-variable*, perception-variable*) >
<!ATTLIST goal
            element-name    NMTOKEN #REQUIRED
            mode            (true|false) #REQUIRED
            codelet-names  IDREFS  #IMPLIED
            >
```

```
<!ELEMENT drive (alias*, attncodelet*, precondition+)>
<!ATTLIST drive
            element-name    NMTOKEN #REQUIRED
            importance      NMTOKEN #REQUIRED
            intensity       NMTOKEN #IMPLIED
            mode            (true|false) #REQUIRED
            codelet-names  IDREFS  #IMPLIED
            >

<!-- behavior, expectation, and behavioral-attention codelets are
defined here -->
<!ELEMENT bcodelet (description*, precondition*, (addition |
deletion)*, variable*, motor-variable*, perception-variable* )>
<!ATTLIST bcodelet
                element-name    ID #REQUIRED
                class-name      NMTOKEN #IMPLIED
                relevant-info   NMTOKEN #IMPLIED
                order-id        NMTOKEN #IMPLIED
                context         NMTOKEN #IMPLIED
                effort          NMTOKEN #IMPLIED
                >

<!ELEMENT expcodelet (description*, precondition*, (addition* |
deletion*), variable*, motor-variable*, perception-variable*)>
<!ATTLIST expcodelet
                element-name    ID #REQUIRED
                class-name      NMTOKEN #IMPLIED
                order-id        NMTOKEN #IMPLIED
                context         NMTOKEN #IMPLIED
                effort          NMTOKEN #IMPLIED
                >

<!ELEMENT attncodelet (description*, precondition*, variable*, motor-
variable*, perception-variable*)>
<!ATTLIST attncodelet
                element-name    ID #REQUIRED
                class-name      NMTOKEN #IMPLIED
                attention-to    (cause | effect | all) #IMPLIED
                context         NMTOKEN #IMPLIED
                effort          NMTOKEN #IMPLIED
                >

<!-- primitive-codelet is defined here, before this was used as
behavior-priming codelet -->
<!ELEMENT bpcodelet (description*, precondition*, (addition* |
deletion*))>
<!ATTLIST bpcodelet
                element-name    ID #REQUIRED
                class-name      NMTOKEN #IMPLIED
                relevant-info   NMTOKEN #IMPLIED
                order-id        NMTOKEN #IMPLIED
                context         NMTOKEN #IMPLIED
                effort          NMTOKEN #IMPLIED
                >
```

```
<!ELEMENT precondition (description*, variable*, motor-variable*,
perception-variable*)>
<!ATTLIST precondition
                element-name NMTOKEN #REQUIRED
                mode    (true|false) #IMPLIED
                >


<!ELEMENT addition (description*, variable*, motor-variable*,
perception-variable*)>
<!ATTLIST addition
                element-name NMTOKEN #REQUIRED
                mode    (true|false) #IMPLIED
                >

<!ELEMENT deletion (description*, variable*, motor-variable*,
perception-variable*)>
<!ATTLIST deletion
                element-name NMTOKEN #REQUIRED
                mode    (true|false) #IMPLIED
                >

<!ELEMENT primingrule (ruleentry+, description*)>
<!ATTLIST primingrule
                element-name NMTOKEN #REQUIRED
                >

<!ELEMENT ruleentry (description*)>
<!ATTLIST ruleentry
                element-name NMTOKEN #REQUIRED
                type        (string | num) #REQUIRED
                value       CDATA #REQUIRED
                filter      NMTOKEN #IMPLIED
                >

<!ELEMENT alias   (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!-- Here I define the structure of attribute and frame types -->
<!ELEMENT basic-attribute-type (description*)>
<!ATTLIST basic-attribute-type
                element-name ID #REQUIRED
                buffer-type    (string | decimal | integer | boolean |
date | time) #REQUIRED
                attribute-value         NMTOKEN #IMPLIED
                attribute-value-range   NMTOKEN #IMPLIED
                attribute-default-value NMTOKEN #IMPLIED
                attribute-filter        NMTOKEN #IMPLIED
                context                 NMTOKEN #IMPLIED
                >

<!ELEMENT text-attribute-type (description*)>
<!ATTLIST text-attribute-type
```

```
                element-name ID #REQUIRED
                buffer-type    (text) #REQUIRED
                attribute-value         CDATA #IMPLIED
                attribute-value-range   CDATA #IMPLIED
                attribute-default-value CDATA #IMPLIED
                attribute-filter        CDATA #IMPLIED
                context                 NMTOKEN #IMPLIED
                >

<!-- sub-structure is eqivalent to say attribute is a relationship of
"hasA" to a sub-structure. -->
<!-- super-stracture means that attribute is a relationship of "isA" to
a super-structure. -->
<!ELEMENT structure-referrence-attribute-type (description*)>
<!ATTLIST structure-referrence-attribute-type
                element-name ID #REQUIRED
                buffer-type    (isA | hasA | associatedWith) #REQUIRED
                attribute-value         IDREF #IMPLIED
                attribute-value-range   NMTOKEN #IMPLIED
                attribute-default-value NMTOKEN #IMPLIED
                attribute-filter        NMTOKEN #IMPLIED
                context                 NMTOKEN #IMPLIED
                >

<!ELEMENT attribute-type (description*)>
<!ATTLIST attribute-type
                element-name    NMTOKEN #REQUIRED
                type            IDREF #REQUIRED
                mandatory       (true | false) #REQUIRED
                >

<!ELEMENT structure-type (description*, (attribute-type* | basic-
attribute-type* | text-attribute-type* | structure-refernce-attribute-
type*) ) >
<!ATTLIST structure-type
            element-name ID #REQUIRED
            buffer-type (flat | hierarchy) #IMPLIED
            context          NMTOKEN #IMPLIED
            >

<!-- Here I declare attribute and frame defining structures  -->
<!ELEMENT basic-attribute (description*)>
<!ATTLIST basic-attribute
                element-name            NMTOKEN #REQUIRED
                type                    IDREF #REQUIRED
                buffer-type             NMTOKEN #IMPLIED
                attribute-value         NMTOKEN #IMPLIED
                attribute-value-range   NMTOKEN #IMPLIED
                attribute-default-value NMTOKEN #IMPLIED
                attribute-filter        NMTOKEN #IMPLIED
                context                 NMTOKEN #IMPLIED
                sketch-pointer            NMTOKEN #IMPLIED
                >
```

```
<!ELEMENT text-attribute (description*)>
<!ATTLIST text-attribute
                element-name            NMTOKEN #REQUIRED
                type                    IDREF #REQUIRED
                buffer-type             NMTOKEN #IMPLIED
                attribute-value         CDATA #IMPLIED
                attribute-value-range   CDATA #IMPLIED
                attribute-default-value CDATA #IMPLIED
                attribute-filter        CDATA #IMPLIED
                context                 NMTOKEN #IMPLIED
                sketch-pointer          NMTOKEN #IMPLIED
                >

<!-- sub-structure is eqivalent to say attribute is a relationship of
"hasA" to a sub-structure. -->
<!-- super-stracture means that attribute is a relationship of "isA" to
a super-structure. -->
<!ELEMENT structure-referrence-attribute (description*)>
<!ATTLIST structure-referrence-attribute
                element-name            NMTOKEN #REQUIRED
                type                    IDREF #REQUIRED
                buffer-type             NMTOKEN #IMPLIED
                attribute-value         IDREF #IMPLIED
                attribute-value-range   IDREFS #IMPLIED
                attribute-default-value NMTOKEN #IMPLIED
                attribute-filter        NMTOKEN #IMPLIED
                context                 NMTOKEN #IMPLIED
                sketch-pointer          NMTOKEN #IMPLIED
                >

<!ELEMENT structure (description*, (basic-attribute* | text-attribute*
| structure-refernce-attribute*) ) >
<!ATTLIST structure
            element-name    NMTOKEN #REQUIRED
            type            IDREF #REQUIRED
            buffer-type     NMTOKEN #IMPLIED
            context         NMTOKEN #IMPLIED
            >

<!ELEMENT variable (description* ) >
<!ATTLIST variable
            element-name    NMTOKEN #REQUIRED
            type            IDREF #REQUIRED
            value-pointer   NMTOKEN #IMPLIED
            context         NMTOKEN #IMPLIED
            >

<!-- Here variable and motor/perception buffer staructures are defined
-->
<!ELEMENT motor-variable (description* ) >
<!ATTLIST motor-variable
            element-name    NMTOKEN #REQUIRED
            act-on          NMTOKEN #IMPLIED
            object          NMTOKEN #IMPLIED
            context         NMTOKEN #IMPLIED
            >

<!ELEMENT perception-variable (description* ) >
<!ATTLIST perception-variable
            element-name    NMTOKEN #REQUIRED
            attend-to       NMTOKEN #IMPLIED
            sketch            NMTOKEN #IMPLIED
            context         NMTOKEN #IMPLIED
            >

<!ELEMENT motor-buffer (description*, motor-variable* ) >
<!ATTLIST motor-buffer
            element-name    ID #REQUIRED
            act-on          NMTOKEN #IMPLIED
            sharing-count   NMTOKEN #IMPLIED
            context         NMTOKEN #IMPLIED
            codlet-names    IDREFS  #IMPLIED
            >

<!ELEMENT perception-buffer (description*, perception-variable* ) >
<!ATTLIST perception-buffer
            element-name    ID #REQUIRED
            attend-to       NMTOKEN #IMPLIED
            sharing-count   NMTOKEN #REQUIRED
            context         NMTOKEN #IMPLIED
            codlet-names    IDREFS  #IMPLIED
            >


<!ELEMENT dm-type (basic-attribute-type* | text-attribute-type* |
structure-referrence-attribute-type* | structure-type*) >
<!ATTLIST dm-type
            element-name    NMTOKEN #REQUIRED
            comment CDATA #IMPLIED
            >

<!ELEMENT dm-instance (basic-attribute* | text-attribute* | structure-
referrence-attribute* | structure* | motor-buffer* | perception-
buffer*) >
<!ATTLIST dm-instance
            element-name    NMTOKEN #REQUIRED
            comment CDATA #IMPLIED
            >

<!ELEMENT general-info (mechanism-developer-info*, application-
developer-info* ) >
<!ATTLIST general-info
            version-name    NMTOKEN #REQUIRED
            version-number  NMTOKEN #IMPLIED
            release-date    NMTOKEN #IMPLIED
            >

<!ELEMENT mechanism-developer-info (dev-memebr* ) >
```

```
<!ELEMENT application-developer-info (dev-member* ) >
<!ATTLIST
            company-name    CDATA #IMPLIED
            group-name      CDATA #IMPLIED
            version-number  NMTOKEN #IMPLIED
            application-description CDATA #IMPLIED
            >

<!ELEMENT dev-memeber (full-name, affiliation, address, email,
contribution* ) >
<!ELEMENT full-name #PCDATA)>
<!ELEMENT affiliation #PCDATA)>
<!ELEMENT address #PCDATA)>
<!ELEMENT email #PCDATA)>
<!ELEMENT contribution #PCDATA)>
```

B BNDL Source Code - Outline of Behavior Streams for the Warehouse Robot Controller

The XML formatted BNDL source code of the behavior net for the warehouse robot controller is very large (over 100 pages) and we could not reproduce it in this document completely. The following gives the outline of the specification that shows the dynamic object modules (DOMs), which include behavior streams and drives, appearing at the root of the XML hierarchy.

```
 <?xml version="1.0" encoding="us-ascii" ?>
<!-- Root element of behaviornet -->
<!DOCTYPE behaviornet (View Source for full doctype...)>
- <!--
theta = the max threshold to become active,
pi = the mean level of activation of a behavior,
phi = amount of environmental activation energy injected per
gamma = amount of activation energy injected by goals,
delta = amount of activation energy taken away by protected
Before 29May05:
theta="60.0"
gamma="30.0"
pi="150.0"
delta="2.0"
phi="15.0"
decay-rate=".05"
-->
- <behaviornet element-name="Warehouse-Robot" theta="35.0"
gamma="50.0" pi="20.0" delta="20.0" phi="20.0" decay-rate=".05">
+ <stream element-name="Unloading-Item1">
<!-- Definition of Unload item1 Stream is done -->
<!-- Stream to shelve item1 -->
+ <stream element-name="Shelving-Item1">
<!-- Definition of Shelve item1 Stream is done -->
<!-- Stream to unload item2 -->
+ <stream element-name="Unloading-Item2">
```

```
<!-- Definition of Unload item2 Stream is done -->
<!-- Definition of Stream to shelve item2 starts -->
+ <stream element-name="Shelving-Item2">
<!-- Definition of Shelve item2 Stream is done -->
<!-- Definition of UnShelve item1 Stream starts -->
+ <stream element-name="Unshelving-Item1">
<!-- Definition of Unshelve item1 Stream is done -->
<!-- Definition of Stream to ship Order1 starts -->
+ <stream element-name="Shipping-Order1">
<!-- Definition of Ship Order1 Stream is done -->
<!-- Definition of UnShelve item2 Stream starts -->
+ <stream element-name="Unshelving-Item2">
<!-- Definition of Unshelve item2 Stream is done -->
<!-- Definition of Stream to ship Order2 starts -->
+ <stream element-name="Shipping-Order2">
<!-- Definition of Ship Order2 Stream is done -->
<!-- Stream to charge robot -->
+ <stream element-name="ChargeRobot">
<!-- Definition of charging Stream is done -->
+ <drive element-name="Like-Unloading1" importance="0.99"
mode="true">
+ <drive element-name="Like-Shelving1" importance="0.99"
mode="true">
+ <drive element-name="Like-Unloading2" importance="0.99"
mode="true">
+ <drive element-name="Like-Shelving2" importance="0.99"
mode="true">
+ <drive element-name="Like-Unshelving1" importance="0.99"
mode="true">
+ <drive element-name="Like-Unshelving2" importance="0.99"
mode="true">
+ <drive element-name="Like-ProcessingOrder1" importance="0.99"
mode="true">
+ <drive element-name="Like-ProcessingOrder2" importance="0.99"
mode="true">
+ <drive element-name="Needs-Energy" importance="0.99"
mode="true">
<!-- One test behavior codelet under the BehaviorNet element
-->
+ <dm-type element-name="dm-type-record" comment="dm memory
types declaration">
<dm-instance element-name="dm-instance-record" comment="dm
memory declaration" />
</behaviornet>
```

C NRPS Behavior Stream Details and Algorithm of Its Planner Behavior

C.1 Adopted Partial-Order Planner Algorithm

A variation of a partial order planning (POP) algorithm shown figure C-1 can be used to implement the task of the *Planner* behavior of the Non-Routine Problem Solving (NRPS) stream discussed in chapter 6. The planner function is called with three arguments. (i) A plan obtained from previous planning step in the on going problem solving process. If there was no previous planning step, plan is initialized to have START (set EFFECTS (START) to the initial state) and FINISH (set PRECONDITIONS (FINISH) to the goal state) actions, ordering constraints list with item START before FINISH (START $\prec$ FINISH), empty variable binding constraints set and empty causal links list. (ii) A contentious subgoal initialized to one of the open preconditions or an unachieved items in the goal state. (iii) A set of actions (behavior codelets) that becomes relevant to the initial problem description.

To shortly define the plan components: (i) ordering constraints' set – each element is of the form Y $\prec$ Z and which means that action Y should be executed some time before action Z; (ii) variable binding constraints' set – each element is of the form "v = x" where "v" is the variable and "x' is a constant or another variable; (iii) causal links' set – each element is of the form Y $\xrightarrow{c}$ Z which means that action of Y achieves proposition "c" for action Z. The ordering constraints must specify a proper partial order - no cycle (for instance, $Y \prec Z$ and $Z \prec Y$ create a cycle) in the ordering constraints and no plan actions raise conflicts with all the causal links. In non-routine problem solving mechanism behavior codelets, as action units, could be recruited as a planning resource with variables in their precondition and postcondition lists to be partially bound. In a given problem solving process, the planner need to keep track of binding lists and should perform timely unification operations. Binding constrains are violated if "v = A" and "v = B" holds for two constants A and B, where $A \neq B$. Although out of scope for our discussion here, there is more to planning in general and POP in particular; for further details, a book by Russell and Norvig (1995) is a good reference.

The planning process continues in the loop until the planner returns with success or failure. It fails either due to unavailability of relevant actions or due to failure of consistency checking of the plan. A plan is consistent if nó violation of ordering or binding constraints. To complete the NRPS process (Chap. 6), the planner could be called multiple times, each time with additional actions available to it. A plan is a solution if there are no open preconditions left. In the plan search, the algorithm does not have to backtrack; this is due to that fact that partial order planner has to remove all open preconditions (unachieved subgoals) and thus could pick them in an arbitrary order.

C.2 Specification of NRPS Behavior Stream

The specification of the NRPS behavior stream, discussed in chapter six, is shown in figure C-2. It has seven behaviors and two goals with a problem-solving and/or curiosity drive as a primary motivator.

```
Function Partail-Order-Planner (Plan, Initial, Goal, Actions)
Returns plan, contentious-subgoal, error-explanation
  Loop do
    Check if plan is a solution; if so, then
      Contentious-subgoal ← null.
      Error-explanation ← "Success"
      Return plan, contentious-subgoal, error-explanation
    Pick a plan action $ACT_{need}$ from ACTIONS(plan) with open precondition (not achieved by an action in the plan) C; first iteration picks the contentious-goal passed as an argument.
    Choose an action $ACT_{add}$ from Action or ACTIONS(plan) that has C as an effect and that u = UNIFY(C, $ACT_{add}$ , BINDINGS(Plan))
    If there is no such step, then
      Error-explanation <- "Failure to find plan – no action in the available set of actions that can satisfy subgoal C".
      Contentious-subgoal ← open precondition C.
      Retrun plan, contentious-subgoal, error-explanation
     Add causal link $ACT_{add} \xrightarrow{C} ACT_{need}$ to CAUSAL-LINKS (plan)
    Add the ordering constraints $ACT_{add} \prec ACT_{need}$ to ORDERINGS (plan)
    If $ACT_{add}$ is a newly added action (FROM Actions) to the plan, then
      Add $ACT_{add}$ to ACTIONS (plan)
      Add START $\prec ACT_{add}$ and $ACT_{add} \prec$ FINISH to ORDERINGS (plan)
    For each $ACT_{conflict}$ that conflicts a causal link $ACT_i \xrightarrow{C} ACT_j$ in CAUSAL-LINKS (plan) do
     For each K in Effects of $ACT_{conflict}$ do
      If SUB(BINDINGS(Plan), C) = SUB(BINDINGS(Plan), $\neg K$) then
       Either promote: Add $ACT_{conflict} \prec ACT_i$ to ORDERINGS (plan)
       Or demote: $ACT_j \prec ACT_{conflict}$ to ORDERINGS (plan)
       If not CONSISTENT (plan), then
         Error-explanation ← "Failure to find plan – consistency check failed with the current plan.
         Contentious-subgoal ← open precondition C.
         Return plan, contentious-subgoal, error-explanation
```

Figure C-1: Partial order planning (POP) algorithm adapted to be used as the planner behavior for the NRPS behavior stream.

Behavior: Initialize Problem Solving Precondition: Problem description ready Precondition: Solving not started Addition: Initial state ready Addition: Goal state ready Addition: Initial action set ready	Behavior: Initialize Planner Precondition: Initial state ready Precondition: Goal state ready Precondition: Solving not started Precondition: Initial action set ready Addition: Deliberated subgoal list initialized Addition: Contentious subgoal ready Addition: Plan ready Addition: Action set ready Addition: Ready to start solving Deletion: Solving not started
Behavior: Planner Precondition: Plan ready Precondition: Actions set ready Precondition: Contentious subgoal ready Precondition: Ready to start solving Addition: Plan ready Addition: Contentious subgoal ready Addition: Result explanation ready Addition: Ready for evaluation Deletion: Ready to start solving	Behavior: Evaluate Solution Precondition: Deliberated subgoal list initialized : subgoals Precondition: Contentious subgoal ready Precondition: Ready for evaluation Addition: Deliberated subgoal list initialized : new list Addition: Solving process in progress ; to goals/drives Deletion: Ready for evaluation
Behavior: Continue Search Precondition: Flagged to continue solving ; priming BC Precondition: Actions set ready Addition: Actions set ready ; old + new Addition: Ready to start solving Addition: Solving process in progress ; to goals/drives Deletion: Flagged to continue solving	Behavior: Terminate with Failure Precondition: Result explanation ready Precondition: Flagged for failure ; priming BC Precondition: Plan ready Addition: Failure to find solution Addition: Solving process done Deletion: Flagged for failure
Behavior: Terminate with Success Precondition: Flagged for success ; priming BC Precondition: Plan ready ; solution Addition: Solution ready Addition: Solving process done Deletion: Flagged for success	Goal: Continue Solving Precondition: Solving process in progress Addition: Like solving Goal: Result Reported Precondition: Solving process done Addition: Like solving Drive: Curious for solution Precondition: Like solving

Figure C-2: Specification of the NRPS behavior stream. Drives are not part of the stream; they pass motivational activation to competencies that satisfy their precondition literals.

D QGOMS Analysis for Warehouse Domain Robot Tasks

```
C:\QGOMS30\BNROBOTF.GMS

GlobalVariableName Value GlobalVariableName Value GlobalVariableName Value

chrgRbtL 9.6 chrgRbtM 10.8 chrgRbtU 12.
dropItemL 7.2 dropItemM 10. dropItemU 14.4
mentActL 0.1 mentActM 0.1 mentActU 0.1
pickupItmL 13.2 pickupItmM 24.6 pickupItmU 38.4
toChrEryL 14.4 toChrEryM 37.6 toChrExtL 36.
toChrExtM 51.6 toChrExtU 67.2 toChrgPtL 14.4
toChrgPtM 31.2 toChrgPtU 48. toCntBlkL 0.
toCntBlkM 68.4 toCntBlkU 115.2 toCntEryL 14.4
toCntEryM 57.6 toCntEryU 105.6 toCntExtL 122.4
toCntExtM 178.8 toCntExtU 211.2 toEndBlkL 0.
toEndBlkM 104.8 toEndBlkU 254.4 toEndEryL 36.
toEndEryM 61.2 toEndEryU 67.2 toEndExtL 120.
toEndExtM 162.4 toEndExtU 192. toShlvPtL 86.4
toShlvPtM 110. toShlvPtU 144. toUnldPtL 43.2
toUnldPtM 100.1 toUnldPtU 144. toChrEryU 50.4

******************************************************************************************************************
title TL=taskLearned D=disabled SC=singleChild timeMin timeAve TimeMax repeatMin repeatAve repeatMax ProbAve LearningTime

TaskInstance T:992.|2021.|3029. R:| |1.00| | Prob:0.00 LTime: 5460
| UnloadAndShelve T:451.|886.|1295. R:| |1.00| | Prob:0.00 LTime: 2250
| | UnloadItem T:216.|457.|700. R:| |1.00| | Prob:0.00 LTime: 1110
| | | GetUnldArea T:| |0.60| | R:| |1.00| | Prob:0.00 LTime: 150
| | | | ThinkOfItemType T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToUnldBlock T:0.60|105.|255. R:| |1.00| | Prob:0.00 LTime: 180
| | | | RecallUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermUnldBlockArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToUnldBlockUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndBlkL|toEndBlkM|toEndBlkU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToUnldEntry T:36.8|62.0|68.0 R:| |1.00| | Prob:0.00 LTime: 240
| | | | Verify-RbtAtUnldBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RecallUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermEntryPtToUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndEryL|toEndEryM|toEndEryU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | SelectItemToUnld T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | LocateItemToUnld T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainLocatedItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToItemToUnld T:43.7|101.|145. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallLocatedItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toUnldPtL|toUnldPtM|toUnldPtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | pickupItemToUnld T:13.6|25.0|38.8 R:| |1.00| | Prob:0.00 LTime: 120
| | | | RbtCmd-PickupItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:pickupItmL|pickupItmM|pickupItmU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToExitUnldArea T:120.|163.|192. R:| |1.00| | Prob:0.00 LTime: 120
| | | | RecallUnldEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-ExitUnldArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndExtL|toEndExtM|toEndExtU R:| |1.00| | Prob:0.00 LTime: 30
| | ShelveItem T:234.|429.|594. R:| |1.00| | Prob:0.00 LTime: 1110
| | | getShlvArea T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | DetermShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShlvBlock T:0.60|69.0|116. R:| |1.00| | Prob:0.00 LTime: 180
| | | | RecallShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermShlvBlockArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShlvBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntBlkL|toCntBlkM|toCntBlkU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShlvEntry T:15.2|58.4|106. R:| |1.00| | Prob:0.00 LTime: 240
| | | | Verify-RbtAtShlvBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RecallShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermEntryPtToShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShlvEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntEryL|toCntEryM|toCntEryU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | selectShlvPoint T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | DetermShlvPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainShlvPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShlvPoint T:86.9|111.|145. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallShlvPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShlvPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toShlvPtL|toShlvPtM|toShlvPtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| | | dropdownItemToShlv T:7.60|10.4|14.8 R:| |1.00| | Prob:0.00 LTime: 120
| | | | RbtCmd-DropdownItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:dropItemL|dropItemM|dropItemU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToExitShlvArea T:123.|179.|212. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallShlvEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-ExitShlvArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntExtL|toCntExtM|toCntExtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermArea T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| UnshelveAndShipOrder T:463.|895.|1299. R:| |1.00| | Prob:0.00 LTime: 2250
| | UnshelveItemForOrders T:240.|443.|618. R:| |1.00| | Prob:0.00 LTime: 1110
| | | getUnshelveArea T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | DetermUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToUnshelveBlock T:0.60|69.0|116. R:| |1.00| | Prob:0.00 LTime: 180
| | | | RecallUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermUnshelveBlockArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToUnshelveBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntBlkL|toCntBlkM|toCntBlkU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToUnshelveEntry T:15.2|58.4|106. R:| |1.00| | Prob:0.00 LTime: 240
| | | | Verify-RbtAtUnshelveBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RecallUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermEntryPtToUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntEryL|toCntEryM|toCntEryU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | selectItemToUnshelve T:| |0.40| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | LocateItemToUnshelve T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainLocatedItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToItemToUnshelve T:86.9|111.|145. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallSelectedItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toShlvPtL|toShlvPtM|toShlvPtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | pickupItemToUnshelve T:13.6|25.0|38.8 R:| |1.00| | Prob:0.00 LTime: 120
| | | | RbtCmd-PickupItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:pickupItmL|pickupItmM|pickupItmU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToExitUnshelveArea T:123.|179.|212. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallUnshelveEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-ExitUnshelveArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toCntExtL|toCntExtM|toCntExtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | ShipItemOfOrder T:222.|452.|681. R:| |1.00| | Prob:0.00 LTime: 1110
| | | getShipAreaForOrderedItem T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | DetermShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShipBlock T:0.60|105.|255. R:| |1.00| | Prob:0.00 LTime: 180
| | | | RecallShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermShipBlockArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShipBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndBlkL|toEndBlkM|toEndBlkU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShipEntry T:36.8|62.0|68.0 R:| |1.00| | Prob:0.00 LTime: 240
| | | | Verify-RbtAtShipBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RecallShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermEntryToShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShipEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndEryL|toEndEryM|toEndEryU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoaSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | selectShipPoint T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | | DetermShipPoint T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RetainShipPoint T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToShipPoint T:43.7|101.|145. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallShipPoint T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-MvToShipPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toUnldPtL|toUnldPtM|toUnldPtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:| |mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | dropdownItemToShip T:| |19.6|19.6 R:| |1.00| | Prob:0.00 LTime: 120
| | | | RbtCmd-DropdownItem T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:dropItemL|dropItemM|dropItemU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | mvToExitShipArea T:120.|163.|192. R:| |1.00| | Prob:0.00 LTime: 150
| | | | RecallEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | RbtCmd-ExitShipArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | | Wait-RbtResponse T:toEndExtL|toEndExtM|toEndExtU R:| |1.00| | Prob:0.00 LTime: 30
| | | | DetermArea T:mentActU|mentActM|mentActL R:| |1.00| | Prob:0.00 LTime: 30
| ChargeRobot T:77.7|239.|435. R:| |1.00| | Prob:0.00 LTime: 930
| | getNearestChargStation T:| |0.50| | R:| |1.00| | Prob:0.00 LTime: 120
| | | DetermNearestChargeStation T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | RetainChargeStation T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | mvToChargeBlock T:0.60|105.|255. R:| |1.00| | Prob:0.00 LTime: 180
| | | RecallChargeStation T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | DetermChargeBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | RbtCmd-MvToChargeBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | Wait-RbtResponse T:toEndBlkL|toEndBlkM|toEndBlkU R:| |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | mvToChargeEntry T:15.1|38.3|51.1 R:| |1.00| | Prob:0.00 LTime: 210
| | | Verify-RbtAtChargeBlock T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | RecallChargeArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | DetermEntryPtToChargeArea T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | RbtCmd-MvToChargeEntryPt T:mentActL|mentActM|mentActU R:| |1.00| | Prob:0.00 LTime: 30
| | | Wait-RbtResponse T:toChrEryL|toChrEryM|toChrEryU R:| |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | mvToChargePoint T:14.9|31.7|48.5 R: | |1.00| | Prob:0.00 LTime: 150
| | | DetermChargePoint T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | | RbtCmd-MvToChargePt T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | | Wait-RbtResponse T:toChrgPtL|toChrgPtM|toChrgPtU R: | |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | doCharge T:10.0|11.2|12.4 R: | |1.00| | Prob:0.00 LTime: 120
| | | RbtCmd-Charge T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | | Wait-RbtResponse T:chrgRbtL|chrgRbtM|chrgRbtU R: | |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | mvToExitChargeArea T:36.4|52.0|67.6 R: | |1.00| | Prob:0.00 LTime: 120
| | | RbtCmd-ExitChrgArea T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
| | | Wait-RbtResponse T:toChrExtL|toChrExtM|toChrExtU R: | |1.00| | Prob:0.00 LTime: 30
| | | Verify-GoalSatStatus T:mentActL|mentActM|mentActU R: | |1.00| | Prob:0.00 LTime: 30
```