A REINFORCEMENT LEARNING FRAMEWORK FOR THE AUTOMATION OF ENGINEERING DECISIONS IN COMPLEX SYSTEMS

A Dissertation
Presented to
The Academic Faculty

by

Arun Ramamurthy

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2019

A REINFORCEMENT LEARNING FRAMEWORK FOR THE AUTOMATION OF ENGINEERING DECISIONS IN COMPLEX SYSTEMS

Approved by:

Dr. Dimitri N. Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Le Song
College of Computing
Georgia Institute of Technology

Dr. Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Simón I. Briceño
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Graeme J. Kennedy
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Frédéric Villeneuve
Digital Innovation
Siemens Energy, Inc.

Date Approved: 3rd December 2018

ACKNOWLEDGEMENTS

I would like to start off by thanking my advisor, Prof. Dimitri Mavris not only for his support and guidance over the past six years, but also for giving me the opportunity to study at Georgia Tech. His willingness to accept me into the Aerospace Systems Design Laboratory a little over 6 years ago exposed me to concepts and ideas that were far beyond the small little bubble of my past. I would also like to extend my gratitude to the members of my thesis committee, in particular to Dr. Simón Briceño and Dr. Frédéric Villeneuve for guiding me through the intricacies of the research work over the past three years. Finally, I extend my gratitude to Prof. Daniel Schrage, Prof. Le Song and Prof. Graeme Kennedy for stepping in to guide me through the final stretches of the thesis work.

At the risk of forgetting someone, I would rather not name anyone.

I would like to thank my colleagues at the Aerospace Systems Design Laboratory with whom I have spent the most enjoyable time of my life. I express my gratitude to my colleagues at Siemens Energy who piqued my interest in the field of machine learning and artificial intelligence. I would also like to thank my colleagues at Siemens Corporation, Corporate Technology, with who I look forward to pushing the boundaries of science and technology.

A special thanks to my parents and family for their extended patience and continuous support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The process of engineering design is characterized by a series of decisions that determine the performance of the final product. Engineers are faced with decisions, such as choices pertaining to the type of model to be used, appropriate parameter settings, system architecture etc., all through the design process and these decisions are undertaken with a desired goal in mind. The decisions themselves manifest as a planned set of actions that are informed by observed behavior and domain expertise. The mathematical formalization of such a design process would resemble that of a sequential decision process.

It is natural to ponder if the underlying logic behind these decisions can be abstracted into a computer program such that, when faced with a similar situation, an intelligent system can aid the ensuing design process. To satisfy this need, expert systems, capable of incorporating design expertise and domain knowledge, have been designed. The state-of-the-art for such systems view them as static entities that are configured to operate on a predefined problem using some variant of rule-based or case-based decision-making methods. The lack of a dynamic quality in the face of evolving design environments and processes necessitates frequent updates and redesign of these systems making their use infrequent in a typical engineering environment.

Recent developments in the field of reinforcement learning have demonstrated significant success in their application to sequential decision-making problems. The reinforcement learning setup of an agent learning from interactions with the environment makes these class of methods a perfect alternative to the static expert systems with

predefined rules. In such scenarios, expert interactions can serve as demonstrations for the learning algorithm and could help train the agent. Further, the exploratory nature of the learning algorithm leads to the possibility that the training agent would identify decision paths that outperform the ones demonstrated by an expert, thereby enabling the system to self-learn to improve the resulting design or process.

The current research work implements a reinforcement learning framework that relies on the principles of life-long learning in order to assist engineering design processes. Assistance is provided in the form of recommendations of design decisions to the design engineer in the course of utilization of the design environment for a given problem. The framework implements aspects of machine learning such as imitation learning from human demonstrations in order to train intelligent agents. The life-long learning aspect of the framework enables adaptation of the trained agents to new and incoming data such that both newly explored portions of the design space and new demonstrations from design engineers are incorporated into the decision making model. The exploratory nature of reinforcement learning algorithm enables the possibility of identifying decision paths that are better that the ones demonstrated by design engineers hence enabling the system to self-learn with the goal of improving the resultant design. An adaptive knowledge graph, representing interactions and effects of human actions, is utilized to encode the sequence of states experienced by the design system with each state represents some unique configuration of a design. An automated approach to the creation of the knowledge graph is implemented through the automation of the knowledge extraction and representation processes. The knowledge is then utilized through an imitation learning process which generates recommendations of actions to design engineers.

The framework investigates aspects of the problem of decision making, namely, the mathematical formulation of an engineering design problem in order to enable sequential decision making, the automation of the knowledge extraction and representation processes and the corresponding adaptive encoding of the extracted knowledge and finally, the ability to learn from the extracted knowledge when the knowledge is extracted from multiple different sources of varying expertise. The framework generalizes the process of knowledge-based engineering across multiple different applications through the utilization of an automated knowledge extraction, representation and utilization scheme. A novel adaptive encoding scheme based on computation of tree-isomorphic differences and generation of a natural language representation feeding to a Doc2Vec algorithm is utilized. The generated encoding is utilized in a demonstration-enabled reinforcement learning algorithm that couples capabilities of deep Q-learning from demonstrations and deep deterministic policy gradients from demonstrations with a modified priority experience replay formulation that accounts for the source of the demonstration to enable real-time in-product contextual recommendations for the purpose of engineering design.

The analysis of the implemented framework is carried out on three fronts. First, it is shown that an agent trained on the problem of UAS design is capable of replicating human-like decisions in the presence of demonstrations. Further, it is shown that if a better decision path is available, the exploratory nature of the algorithm enables the identifications of designs that are better than the best demonstrated one. Finally, an analysis of the robustness of the agent to changes in the set of requirements is performed in order to estimate the flexibility of the framework and its capability to generalize across different but similar problems. A rigorous analysis on the impact of training times,

amount of data and the size of the problem is performed in conjunction to the first problem setup. Second, an approach to automate the extraction, representation and utilization of knowledge from multiple sources of information is demonstrated on the problem of automation of engineering systems. Finally, it is shown that the implemented framework outperforms existing state-of-the-art systems that rely on rule-based inference and case-based reasoning. It is shown that the agents trained by the implemented framework are more adaptive to the problem at hand and require less configuration in comparison to the state-of-the-art systems.

CHAPTER 1. INTRODUCTION

The problem-solving capabilities demonstrated by humans are unparalleled in nature. While not unique to humans, researchers have shown a direct correlation in the size of a mammal's brain to its problem-solving capability [1]. Given that human beings boast the largest Encephalization quotient [2] amongst all mammals, the conclusion that our problem-solving capabilities outshine other animals is aptly justified. In fact, it has been argued and recognized that problem-solving is one of the fundamental cognitive process in human beings [3], [4]. While humans rely on experience and knowledge to inform actions related to problems similar to the ones that one may have encountered [5], creativity in humans drives the manner in which unique issues are addressed [6]. This creative outlook to problems, through history, has resulted to some, though intermittent, significant changes in the way human life has been carried out. For example, in comparison to life of our ancestors, there is, in general, a significant difference in the quality of life and the way in which day-to-day activities are carried out. This difference is a result of the creative innovations that have resulted in era-defining solutions which transform human lifestyle. While the impact of these innovations on the quality of life may have been inadvertent, they are without a doubt a result of the capabilities offered by human intelligence. A handful of such examples are: the use of the first stone tools over two million years ago, the development of writing over the five millennia ago, the inception of the concept of medicine around the turn of the common era, to the invention of the steam engine in the year 1712 [7], electricity in the early 19th century [8], penicillin

in the year 1912, transistors in the year 1947 and the world wide web in the year 1989[1]. A trend that can be observed with the progression of these innovations is the transition in the nature of the innovation from the physical to the intellectual. It is, hence, reasonable to assume that future innovations too will push the boundaries of our intellect. Another trend that is evident when the duration between these innovations is analyzed is the increase in the frequency of highly intellectual innovations in the recent past. If one were to define quality of life as an estimate of the comfort and ease with which life is carried out at any instant of time and at a given location, a drastic change to this measure can be observed at instances where human creativity has resulted in an innovation that has attempted to automate some of the menial and tedious tasks associated with daily activities. This can be observed with inventions even before that of the introduction of the printing press in the 1440s [9] to the launch of the first personal computer in the year 1976 [10] and since, a large number of engineering inventions have aimed at addressing the automation problem. Some of the more recent innovations, such as smart phones, robotic agents and intelligent systems, have taken a unique perspective to the problem of automation, one that has never been observed in history. A key aspect common to these newer innovations is the presence of some element of *smartness* or *intelligence*. The realization of the zenith of our current understanding of automation would be the creation of an intelligent systems or artificial agents that would be capable of replicating human-like reasoning, deduction and decision-making, in other words, the creation of artificial intelligence.

[1] The author recognizes that there are several other innovations in other fields such as the discovery of micro-organisms in the 17th century that dictates all of modern medicine, or the development of artificial satellites from the late 1970s that have resulted in the modern global environment. But in order to establish a connection to the rest of the thesis work, these are left out of the discussion both here and in the subsequent Figure 1.

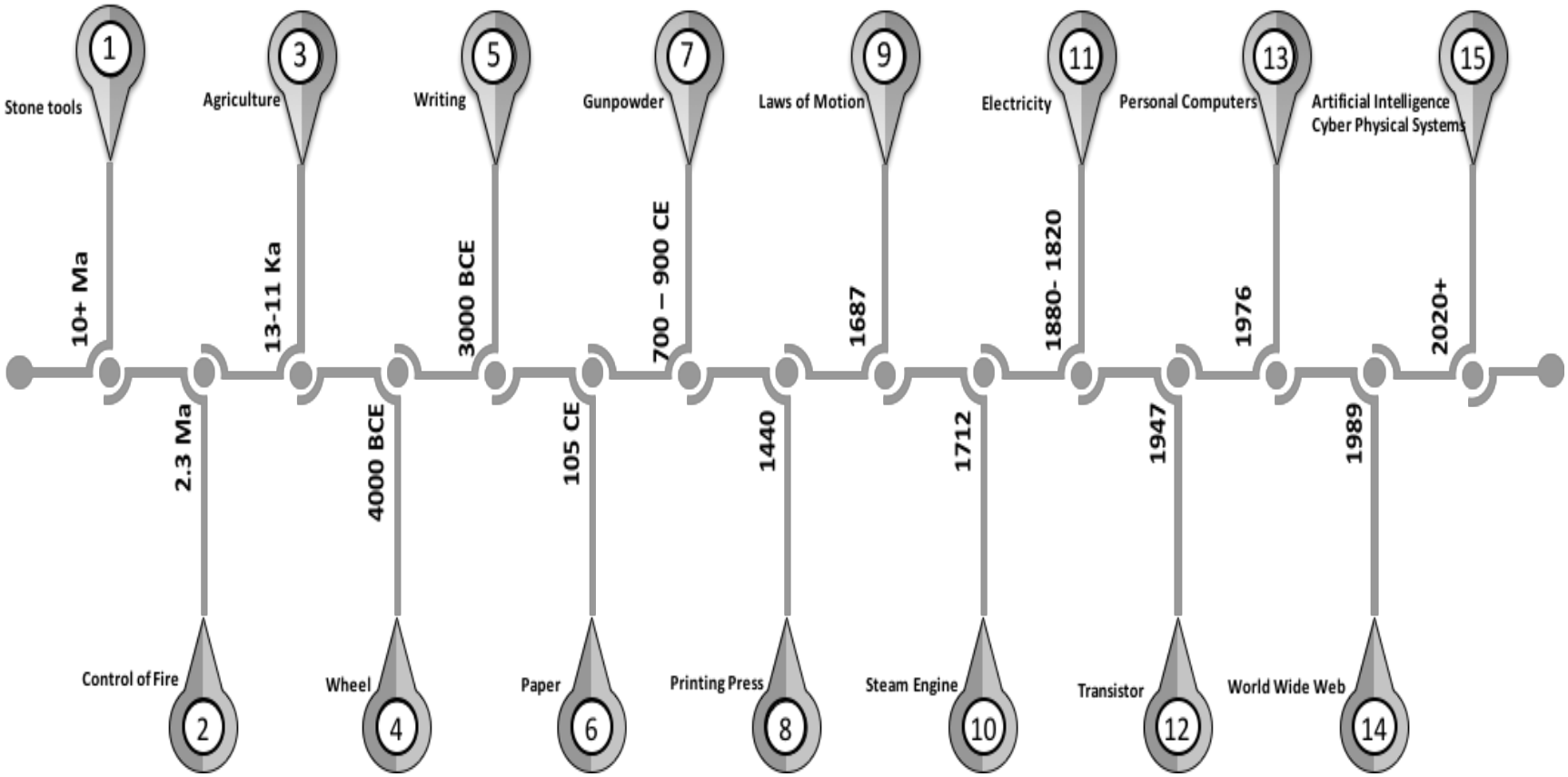


Figure 1.1: A timeline of some of the key innovations through history

Engineering innovations since the 17th century can be categorized into three revolutions that have impacted the manner in which products have been manufactured. The first revolution, more popularly known as the steam revolution or the industrial revolution, was triggered in the early 18th century as a result of the development of the steam powered engine. This in turn lead to the mechanization of the world resulting in the creation of factory floors. The result of this revolution was the transition of production of products from being a household occupation to the factories; greatly increasing the production rate and significantly reducing the cost [11]. The second revolution, often termed the electric revolution, was a result of the introduction of electricity and electrical power to the factory floors. The result of this electrification of the factory floor was the replacement of outdated steam powered machines with newer electrical ones. The second industrial revolution culminated with the introduction of mass-production in the factory floors where a cheaper standardization of products was adopted in place of expensive specialization. Moving through the course of the years, through the first and second-world wars, and arriving at 1947, a year that introduced the transistors. The introduction of transistors altered the human perception of computers, which from being large bulky machines transitioned to entities that are now handheld and more powerful that all of history put together. These developments in the computational capabilities lead us to what is commonly believed to be the third industrial revolution – the digitalization of the world. The digitalization era has seen the wide spread use of computers and computational devices, a result of the increasing computational power and memory and an accompanying reduction in prices, for the purposes of automation, entertainment, and, in general, the day-to-day activity management. These computational systems are so

prevalent in modern society that most people carry on about their daily activities without knowing that there is, in fact, a computational system in the background aiding them in said activity. While these past three industrial revolutions have been an inadvertent result of a set of innovations, humans have not consciously attempted to alter the status quo. In contrast, the several prominent entities [12], [13] around the world are now consciously attempting to trigger the fourth industrial revolution; one that involves automation of the world. This attempt, originally launched as INDUSTRIE4.0 [12], attempts to create a fully connected production plant through the use of cyber-physical agents capable of making autonomous decisions and communicating with other entities through the use of the internet-of-things. A key aspect of each of the characteristics, i.e., cyber-physical systems, autonomous agents and internet-of-things, is the presence of artificial intelligence. This has led to the belief that, as with the replacement of steam power machines with the introduction of electricity during the electrical revolution, the introduction of artificial intelligence will trigger the start of the fourth industrial revolution impacting the manner in which modern digitalized environments are designed and behave in the future.

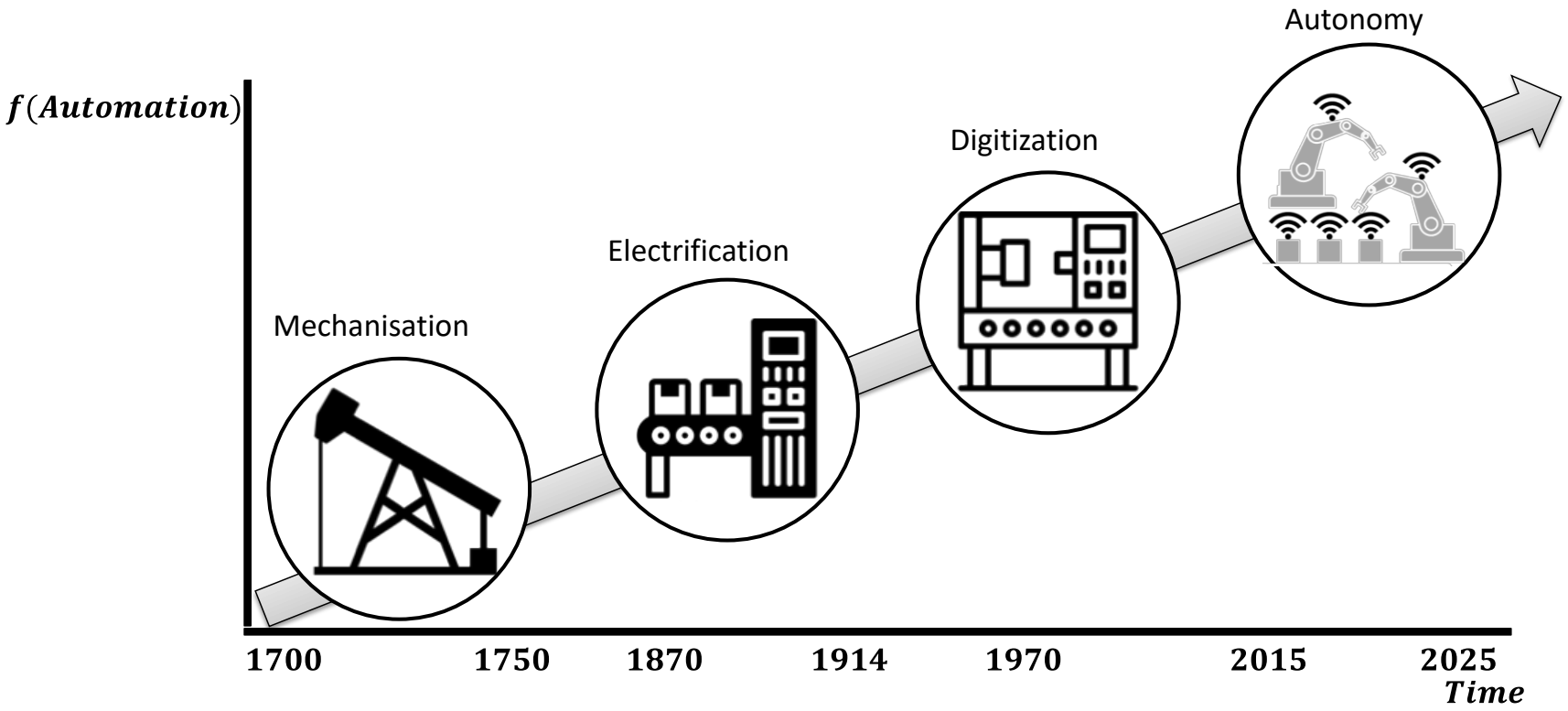


Figure 1.2: The Progression of Industrial Revolutions occurring over the past 300 years

"Artificial Intelligence is the new electricity" – ***Prof. Andrew Ng***

1.1 Artificial Intelligence

Homo-sapiens – literally the "wise man" [14], so called as a result of what is understood to be the distinguishing feature from our ancestors, our *intelligence*. The topic of intelligence is one that has been studied philosophically, for thousands of years. Intelligence has often been seen as a distinguishing characteristic of the human mind that has separated us from other creatures. Thus, historically, philosophers have attempted to understand the nature of human intelligence by pondering on two primary questions, rewritten in general terms as,

- How does the human *mind* work?
- Can non-humans have *"minds"*?

This foray into an attempt at understanding the human mind has been made with a hope of attaining an understanding of intelligence. But, as with every topic, following the natural progression of thought, one ought to first define the term *mind*, paraphrased as [15],

---

Mind

"The element in an individual that enables the activity of feeling, perception and reasoning about one's surroundings."

Using this definition, a computer science perspective to the latter of the questions would lead one to the conclusion that any non-human would be adequately capable of

replicating what is perceived to be the rational portion of the human mind. This rational portion that is dictated by logical reasoning is of utmost importance in one's ability to perceive and understand one's surroundings. Owing to the nature of logical reasoning, it is a just assumption that the rational portion of the human mind can be represented by a machine that is capable of reproducing human intelligence; providing our first glimpse into *artificial intelligence*. But before addressing the topic of artificial intelligence, it is essential to formally define intelligence. Owning to the nature of linguistics, one could assemble an assortment of definitions for the term intelligence, starting off with [16], [17],

---

Intelligence

---

*"The ability to **think** and **understand** things"*

*"The ability to apply **knowledge** to manipulate one's surroundings"*

While these definitions, pedantic as they are, provide an insight into intelligence, they raise a few additional questions by defining intelligence in terms of three additional terms that require further investigation, i.e., thinking, understanding and knowledge. But before the open questions about the pending definitions is addressed, one needs to close the loop on the definition of artificial intelligence. With an understanding of the term intelligence, taking a linguistic perspective, once more, one would arrive at the definition of artificial intelligence as being "a branch of computer science that deals with the simulation of intelligent behavior in computers" [18]. While this definition does provide a general outlook for the term artificial intelligence, it fails to provide any insight into its

nature or characteristics. To accomplish this, one needs to approach the problem of generating a definition from a technical perspective. A definition that is supported by some of the leading technical experts in the field of artificial intelligence can be paraphrased [19] as follows,

---

Artificial Intelligence

---

*"Artificial Intelligence is the study of computations that makes it possible to **perceive**, **reason** and **act** in one's environment."*

This definition of artificial intelligence distinguishes the field from that of philosophy and psychology by emphasizing the computational realization of the capabilities of the intelligence machine and also ensures its distinction from the field of computer science due to the importance placed on the realization of perception, reasoning and action. This, in essence, establishes the field of artificial intelligence as a distinct field in the domain of science and engineering focusing on solving real-world problems by the development of means for the representation and utilization of information gathered from one's environment.

1.2 Knowledge

Before decomposing the definition of artificial intelligence with the hope of identifying its characteristic features, the open-question related to the definitions of thinking, understanding and knowledge have to be addressed. While these are terms used in the English vernacular all around the world, they are often difficult to explicitly define. An analogy to the mathematical world would indicate these terms as being similar to

prime numbers as they are, perhaps, indivisible into their constituent components. But these issues having already been addressed in linguistics which provides us a set of definitions for the terms thinking and understanding, amongst which are [20],

---

Thinking

---

*"The activity of using one's brain to **understand** a problem, making judgements about it to develop a solution."*

This definition of thinking simplifies the considerations to be made as one now has the term thinking defined in terms of understanding. One final linguistic peek – now at the term understanding reveals its definition as definition being [21],

---

Understanding

---

*"The act of possession of **knowledge** about a certain subject."*

which further simplifies the considerations to be made as both thinking and understanding are expressed in terms of one single concept, knowledge. This now leave just one unanswered question, "What is knowledge?". Sadly, the answer to this question is not as easy as the identification of a single definition for the term knowledge. This is due to the difficulties offered by the technical considerations of the term. From a linguistic perspective, there are three terms that are scarcely distinguished in everyday vernacular. But the technical consideration of the term knowledge demands that a

distinction be established between these concepts which results in a confusion regarding their definitions when translated from the vernacular. These concepts are,

- Knowledge
- Information
- Data

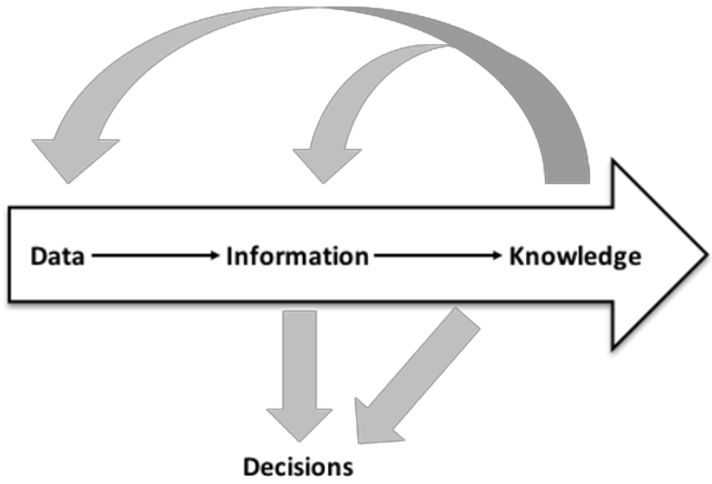


Figure 1.3: Process in which knowledge, data and information interact in the process of decision making [22]

One may develop a distinction in the associated value with each one of these terms, but that hardly serves as a clear definition from the perspective of gaining insight into their characteristics. A more precise distinction granting the necessary insight into the intricacies of the differences between these terms follows from the work of [22]. In addition to the establishing a distinction between the terms themselves, their relationship amongst themselves and to the concept of human intelligence is offered by [22] and is highlighted in Figure 1.3. The process of generating a distinction between the three terms starts with identification of the source from and manner in which the three are gathered. **Data** often comes as the raw representation or observations that can be gathered from

one's environment and is a result of one's perception of the surroundings. These bits of data are things for which the mind has gained an ability to gather, represent and quantify. **Information**, on the other hand, refers to processed data that is a result of execution of one's judgement for the identification of data relevant to a particular context. Information, thus, is a product of an analysis of a large quantity of data resulting in the identification of a set of key bits that are essential for consideration for the problem at hand. Finally, **knowledge** would represent a more generic understanding of the concepts represented by any gathered information enabling its generalization across multiple contexts. This is often a result of experience and typically exists in two forms, *tacit* and *explicit* [23]. Tacit knowledge refers to an understanding of a problem that is innate to a person that is often difficult to transfer, while explicit knowledge refers to the transferrable knowledge that can explicitly specified, written or coded and often relies on one or more bits of tacit knowledge.

1.3 Characteristics of an Intelligent Machine

Having gained an insight into the differences between the terms data, information and knowledge and having defined the terms thinking, understanding and knowledge, one may, once more, return to the identification of the characteristic features of the field of artificial intelligence. To identify the characteristics of artificial intelligence, it serves to first define the goal that is to be served by the development of an artificial intelligence application for an engineering problem. This problem has been addressed before and a key target of artificial intelligence is identified as being [24] "the application of intelligent machines to tasks that requires considerable intelligence from a human operator", i.e., in essence, the development of an intelligent machine. Prof. Minsky, in the

book The Emotion Machine [25], theorizes a process for the development and operation of one such machine. The conclusions reached are based on similarities drawn from observations of human behavior, in particular, infants. Owning to the instinctive nature in which infants make decisions, the machine is termed as an "instinct machine". This machine, illustrated in Figure 1.4, is theorized to comprise of three modules,

- a sensory module, such as the eyes, ears, skin, etc., whose primary function is to perceives both the problem and the state of one's environment
- a knowledge module, which houses a set of relationships that map perceived states to actions
- and, finally, a motor module, which applies actions in response to the problem faced.

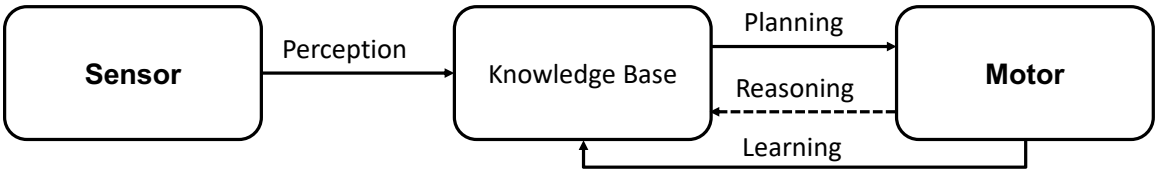


Figure 1.4: A block diagram of Minsky's Instinct Machine

Given this understanding of the instinct machine and based on the definition of artificial intelligence, one can identify a set of five key characteristics for the any artificial agent. These are given as,

1. The machine should have a capability to perceive the environment.
2. Decisions made by the machine should be knowledge-based.
3. The machine should have the capability to plan a course of action based on perceived and expected states of the environment.

4. The machine has to be capable of learning from the perceived value of any action taken.
5. The machine ought to be capable of reasoning to justify the decisions made and the actions taken.

Given that they form an important part of an artificially intelligent machine, this analysis requires one to now define two additional concepts; planning [26] and learning [27].

Planning

"Planning refers to the process of generating a representation of future behaviors of an environment prior to the utilization of a plan in order to constrain or control the behavior of an organism."

Learning

"Learning refers to the process of changing an organism's capacities or behavior through experience."

1.4 History of Research in and Modern Applications of Artificial Intelligence (State-of-the-art in Artificial Intelligence)

Through humble beginnings in the early 1940s to rather complex applications in the recent past the field of artificial intelligence has seen its application to a variety of problems in the field of science and engineering. Though still in its infancy in comparison to fundamental sciences such as physics, chemistry or biology, the field of

artificial intelligence has shown tremendous promise as being the next frontier for science and technology. In fact, the field, along with molecular biology, is cited to be one of the most desired field of research and work by engineers in other fields. Figure 1.5 summarizes some of the key developments that have occurred over the past seventy years in the field of artificial intelligence demonstrating not only a facet of the variety in the application fields, but also the complexity of some of the applications that have been addressed.

Research in the field of artificial intelligence began in the early 1940s with the development of the neuron model [28]. This artificial neuron model was composed of a set of neurons, each activated by the effective stimulus resulting from its neighboring neurons, laying the groundwork for the original neural networks. The model also laid the foundation for trainable networks by suggesting that the network models could learn representations of problems. This particular topic was addressed in the late 1940s with the release of the Hebbian model [29] for the adaptation of neurons in the brain during the learning process. The concept of neural computing was realized with the first neural network computer being built in the year 1950. But it wasn't until the year 1956 that the term artificial intelligence was coined [30] at a workshop at Dartmouth. The period before 1956 that resulted in the inception of the field of artificial intelligence is often termed the "Dark ages of Artificial Intelligence".

The period following the workshop at Dartmouth, was one filled with great expectations from the field of artificial intelligence. In the early 1950s a computational program capable of playing the game Checkers was developed. It was demonstrated to learn and perform significantly better that the creator of the program. This effectively

silenced some of the critics of the field who argued that computational algorithms would be incapable of doing things that they weren't explicitly programmed to. Around the same period the high-level programming language Lisp was designed and developed [31] which quickly established itself as the standard programming language for the purpose of artificial intelligence. The expectations from the field of artificial intelligence were bolstered by developments in artificial intelligence applications such as the General Problem Solver [32] and Geometry Theorem Prover [33] that were capable of imitating problem-solving protocols demonstrated by humans. The period also saw developments in the field of vision, reasoning, and natural language understanding through the late 1960s and early 1970s. One of the key contributions that remains valid even today was the publication on the perceptron convergence theorem in 1962 [34] that forms the basis for the backpropagation algorithm utilized in the training of modern neural networks. But, in the latter half of the 1960s funding to most artificial intelligence research programs was cut in the western world. As a result of this, the developments in the 1970s and 1980s were commercial in nature. Several commercial applications for the purpose of medicine, mining, and molecular analysis were formulated through the use of a heuristic-driven knowledge base [35]–[39]. In 1969 a publication [40] revealed the limitations of the perceptron model, effectively bursting the bubble on the expectations of artificial intelligence.

Following some of the difficulties in the realization of the expectations of the artificial intelligence programs, focus in the 1970s and 1980s were primarily on the commercialization of artificial intelligence applications. The period saw a shift of focus in the means in which artificial intelligence algorithms were developed. Supervised learning algorithms were prioritized over others approaches as these algorithms were

demonstrated to efficiently recognizing patterns in data. These demonstrations further bolstered research efforts in the field of supervised learning. This period of change in the focus of research is often termed the "AI Winter", which, perhaps, ended with the reintroduction of the backpropagation algorithm in the 1980s [41]. At about the same time significant developments were seen in the field of reinforcement learning, a concept originally introduced by Alan Turing in the 1950s. Bolstered by the improvements in computing power, the realization of the marriage between neural networks and reinforcement learning was demonstrated in the year 1994 with the implementation of the IBM's backgammon player [42].

The recent resurgence in the field of artificial intelligence has been guided by the goal to reproduce "human-level AI" and is aided by developments in the neural network training routines and improvements in the computational capabilities. The availability of large amount of training data further bolsters the application of artificial intelligence to new fields. Developments in the field of neural networks, such as the creation of advanced neural network architectures, such as convolution neural networks, recurrent neural networks etc., have propelled the field of artificial intelligence to new applications related to vision, text and speech processing [43]–[46]. The introduction of deep neural network has been another factor that has lent itself to the further the research activities in the field. In particular, the combination of deep neural networks with the methods of reinforcement learning, resulting in deep reinforcement learning, have recently shown to be capable of producing human-like performances on certain tasks. Complementary developments in supervised learning and the underlying numerical optimization routines, too, have been significant contributions to the development of the field. Recent developments have not only included improvements to the training algorithms and

application to new fields, but also investigation of applications to different domains. For example, while in the past human-like AI was primarily applied to problems involving discrete decision making, new algorithms have been developed to enable the application of intelligent machine to problems in the continuous domain [47], [48] or even, mixed discrete-continuous domains.

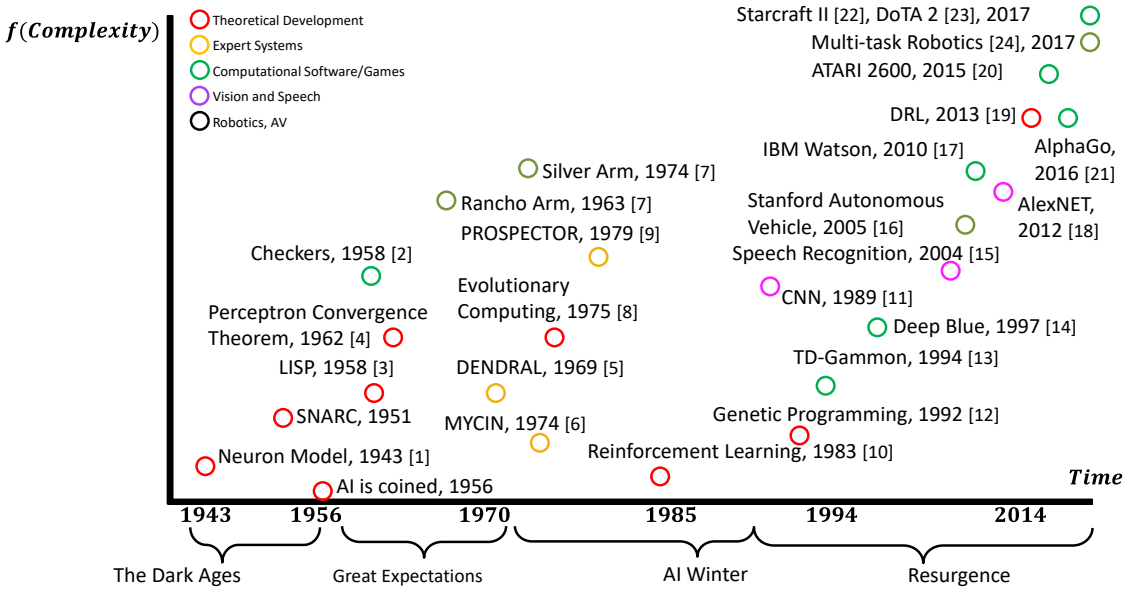


Figure 1.5: Some of the major developments in the field of artificial intelligence since its inception

While the preceding passages briefly summarize some of the key developments observed in the field of artificial intelligence over the past seventy years, it is essential to note that the field is very much in its infancy and new developments are being reported rather frequently, making the estimation of the state-of-the-art for such a field rather difficult. Another factor that makes the establishment of the state-of-the-art difficult for the entire field is the differences in the implementation for each application. For example, the application of end-to-end learning for computer vision would comprise of a different network architecture and training algorithm in comparison to another application such as

self-driving vehicles. But having made these notes, one could argue that the artificial intelligence methods that are enabled through the use of a variety of architectures of deep neural networks and generalized reinforcement learning algorithms for the purpose of end-to-end learning would be an apt representation of the state-of-the-art in the field of artificial intelligence, without any loss of generality.

Based on the information presented in the preceding sections, one could tally up the set of application where the field of artificial intelligence has seen significant application. These would include,

- Autonomous vehicles
- Computer vision
- Logistics and planning
- Gaming
- Natural Language Processing and Speech Recognition
- Robotics
- Expert and Decision support Systems

Prior to addressing the motivation for the thesis work presented in this dissertation, it is essential to gain an understanding of the concepts of a complex system and the complexity in modern engineering design systems. These topics are addressed in the following sections.

1.5 Complex Systems

Before the topic of complex system is addressed, it is essential to develop an understanding of the term *complexity*. To do so an answer to the question, “What is

complexity?" is sought. The concept of complexity varies in both its definition and metrics of consideration depending on the application considered [49], to an extent that there is neither a single science of complexity nor a complexity theory that exists solely for the generalized measurement of the complexity of a certain entity [50]. For example, in the context of computation, complexity, or computational complexity is defined to be the metric of interest and is typically given by the number of operations or time taken to perform a certain computation. But, on the other hand, in the example of information relay, complexity is often represented as the minimum of bits of information necessary to convey a concise description of the entity's regularities. While this interpretation of complexity does not lend itself to comparison across different contexts of problems, it, certainly, is suitable in establishing a means of comparison for problems of the same context. The lack of consensus and, in fact, vast difference in interpretation of complexity in different contexts, necessitates a more abstract perspective of the problem. This abstract view is guided by set of questions posed in the early 2000s [51] that helps quantify the degree of complexity of any entity or process, these questions being,

- How hard is the entity or process to describe?
- How hard is the entity or process to create?
- What is the degree of organization in the entity or process?

Using these guiding considerations, it would now be possible to determine if a given system behaves as a complex system. The analysis of the term "complex system" must being with a consideration of its definition. As with complexity, while there is generally an agreement on the presence of one such entity as a complex system by the members of the scientific community, it is rather difficult to find a consensus on the

definition for one [52]. Perhaps, one of the better definitions for a complex system was offered by Simon [53] that roughly relates the system complexity to the number of interacting parts of the system, the manner and impact of these interactions. In essence, complex systems contain a large number of interacting components that interact in a non-trivial manner such that the resulting system as a whole is greater than the sum of the individual components. Researchers [52] have identified a set of key characteristics to help classify a system as being complex. These characteristics demand,

- a non-linear system
- having a notion of order, i.e., not completely random nor completely deterministic,
- yet robust with some sense of autonomy
- demonstrating emergent behavior
- with a hierarchical organization
- and having feedback of information for each component based on the manner in which the components neighbors interact

Several naturally occurring entities can be classified as being examples satisfying these characteristics of complex system, such as an organism's brain, fluid turbulence, etc. Likewise, several engineered products, too, demonstrate the necessary characteristics to be classified as complex systems, for example, an automobile, a gas turbine, the world wide web etc. A technical definition of a complex system can be given as, paraphrased from [50],

Complex Systems

Complex systems are interdisciplinary entities in which a large number of relatively simple entities organize themselves, without a central controller, to create a whole capable of exhibiting patterns, utilizing information and, also, demonstrating the ability to evolve and learn.

1.5.1 The Aircraft as a Complex System

The manner in which aircrafts are designed have seen a drastic change over the course of the past century. From the inception of flight through to the early 1940s, aircrafts were predominantly designed from a mechanical system and aerodynamic design perspective. Components were designed so as to involve very few interactions, with each being designed to operate in independent silos. The goal of such as design was typically a reduction in system weight to ensure that the generated lift enabled flight. Though over time with improvements in the capabilities offered by the computational resources and as a result of improvements made to the electronics onboard the vehicle, the design paradigm has transitioned towards a multidisciplinary setting. The incorporation of avionics onboard the aircraft, in particular, brought forth with it an increase in the number of interacting components. On the other end of the spectrum, developments in computational capabilities have enabled the consideration of the complexity associated with the physics of flight during the design process. Although the elementary components onboard perhaps still remain simple in nature, the magnitude and nature of their interactions makes the entire aircraft system complex.

1.6 Complexity in Engineering Design Applications

As with engineering products and processes, complexity can exist in the applications used to realize these products and processes. In fact, owing to the growing demand for a flexible and all-encompassing design environment in the modern engineering workplace, modern design applications can often be quite complex in nature. The competitive nature of the modern engineering marketplace bolsters the demand for high efficiency from design teams which is in turn is relayed to demands of flexibility and efficiency from the design applications used. This trend of heightened complexity in the application development process has been observed in the field of enterprise application development [54] and relates directly to the complexity observed in engineering design applications. A key contributor to the complexity is the adopted trend in the development of software applications for the purpose of design engineering. The development of modern engineering design applications is typically driven by a commercial aspect where third-party developers provide the necessary software capabilities to design teams enabling them to accomplish the desired goals. These third-party organizations, though in a bid for market superiority, often attempt to generalize the developed applications so as to be suitable to a variety of fields. While this makes the developed application extremely flexible, in the context of one single application, it also adds an undue burden on the design engineer who now have to overcome the learning curve associated with an extremely flexible new software application. It is often the case that the flexibility of the system is directly proportional to the associated learning curve, which make most modern software rather difficult to use out-of-the-box. Engineering corporations that develop commercial products, while aware of the complexity in modern design system, often prefer to spend the capital necessary to train engineers in the use of

off-the-shelf software rather than the development of a more simplified in-house alternative. This behavior can be attributed to two primary factors,

- The time, effort and expertise required to develop an effective and efficient software application that can be reused across several projects is often quite high. This is evident with an analysis of the annual revenue of ANSYS Inc., which provides solutions to engineering design problems in multiple disciplines. The company boasts an annual revenue of over $1 billion for the year 2017 [55]. Even a modest assumption of 10% of this revenue being fed back into product development would imply a $100 million development expenditure for the software application on a yearly basis. Similar trends are seen by other major players in the engineering software world such as Siemens Digital Factory [56], Dassault Systèmes [57], Mathworks [58] just to name a few.
- The time and costs associated with the verification and validation of any software developed in-house often prove exorbitant for design companies that are driven by the requirements of reduction in design cycle times and faster times to markets. This is a direct result of the competitive nature of the modern market environment.

As a result, in place of a specialized application, that are suited for the problem context, a much more flexible application is utilized as it is typically usable out-of-the-box and adheres to strict validation guidelines.

As the primary function of these design application is to enable the execution of the design process, they need to be able to retain some representation and display of the instantaneous state of the design under consideration. These displayed states guide the

decisions taken by the design engineer during the design process. But owning to the flexibility in these systems, there is a considerable amount of complexity introduced due to the nature of the representation chosen. This is a result of the fact that the detailed representation of the design has to be abstract enough to capture the common features across multiple fields of engineering. This abstraction is typically achieved by relying on high fidelity representation through the use of geometry modeling and detailed physics modeling which is one of the primary causes for the complexity in modern engineering design applications.

1.7 Motivation

The process of engineering design is one that is increasingly being recognized as being characterized by a series of decisions [59] in which design engineers are tasked with making appropriate decisions for the purpose of the creation of the final designed product. Design engineers are faced with decisions at multiple different levels and of multiple different natures, such as choices related to the type of model used, choices related to the settings of different analysis and design parameters, and, also, architectural choices, to name a few. These decisions can be viewed as being made at different levels of abstractions and they are always driven by a certain design goal and scenario in mind. A representative example of the design goal may be given by the task of identification of a set of suitable values for design parameters in an exploratory search scenario where a feasible and viable design is sought, and, on the other hand, an example for a scenario could correspond to analysis performed for the identification of a set of performance parameters. In this process of decision making, the choices made by the design engineer manifest themselves as a sequence of actions. These actions in turn are informed by the

observed instantaneous state of the design, the goal or target in mind and also the domain expertise pertaining to the problem under consideration.

In the modern digital setting, design processes are exercised through the use of design tools that manifest in the form of software applications. Most design application utilize the abstract framework of product design, modeling and simulation as the de facto standard means for a design engineer to interact with the state of a design and its estimate the performance at any instant of time. Given that the design process is characterized by a sequence of decisions that relies on the application of engineering judgement and domain expertise, it is natural to ponder if the underlying patterns in the engineering logic that dictates the decisions made can be identified, captured and abstracted into a computer program such that the program can intelligently support and guide the design engineer when faced with similar design scenarios in the future.

The increasing demand for accuracy in the design simulations results in an increase in the complexity in the design applications. This has in turn resulted in the adoption of a more complex representation for the state of the design through the use of visual representation schemes such as the ones enabled through the use of computer aided design or ones enabled by complex system block diagrams. As a result of increasing complexities in the design applications themselves, design engineers now have to overcome a steep learning curve. This requires both a significant amount of exposure to the design application, achieved by hours of training logged on the application, and also exquisite command of the subject matter which takes years of studies and experience in the field. These factors result in a very long lead time in achieving a mastery over most modern design applications. Another factor that contributes to the difficulty in achieving

mastery over an application is that the expertise gained by one engineer is not readily transferrable to another. While engineers may collaboratively gain expertise in the usage of an application, through trainings and workshops, effectively reducing the number of hours spent, the resultant lead time would certainly not correspond to that of the ideal scenario; where knowledge is transferred from one engineer to another or shared between engineers. A typical example of the collaborative learning setting may involve an *"expert"* engineer training a set of *"novice"* engineers in the intricacies of the application. But, in reality, access to such expertise is often limited, partly due to the lack of pervasive presence of such expert engineers and, in part also, due to the limitations of a typical work environment. Hence, any organization that relies on the utilization of complex design applications would benefit from the computational representation of the design expertise. If such a representation is achievable, then the applications could be implemented in such a way so as to train novice design engineers in place of the expert. Current standards in engineering design applications do implement some aspect of these training routines, typically represented in the form of,

- Design applications enhanced with the user guides and documentations.
- Design applications that have embedded training routines.
- Decision support and expert systems capable of providing real-time recommendations for design engineers.

Traditional design systems that are used in the engineering workplace fall under the first and second categories. Both these categories rely on a predefined set of scenarios that are to be used to educate a design engineer, which of course means that they lack any sense of adaptability. Further as these documentations and training routines are typically

written/developed by application developers, they may not reflect use cases that are of interest to the design engineer. The third category represents a more dynamic system that is utilized to provide recommendations to "users" based on historical data and/or current application state. But these systems are not as prevalent as the other categories in engineering design field and have historically been applied for websites, such as Google, Amazon, etc. and entertainment applications, such as Netflix, Hulu, Facebook, etc.

In light of the limitations of traditional training systems and the lack of the pervasive use of recommender systems in engineering design, the thesis work presented here aim to develop a generic framework that can enable the use of techniques offered by the field of machine learning in order to provide recommendations assisting design engineers in the process of making design decisions. It is sought to base the recommendations on the knowledge extracted during a design engineer's interactions with the design application. As a design engineer interacts with the application, i.e., exercises the design process, all the necessary knowledge associated with the creation of the design would be encoded in some means within the application. This encoded knowledge shall be extracted and utilized to train a learning algorithm such that patterns in engineering decisions can be exploited and recommendations based on these patterns made. Thus, the overarching goal guiding the research work is stated as follows,

---

Research Goal

Develop a methodology, founded on strong mathematical principles, that enables the automatic extraction and representation of design knowledge such that the extracted knowledge can be utilized by a

learning agent to aid, automate or replace design engineers in new, but similar, scenarios.

1.8 Scope of the Research Work

As the current research work attempts to apply techniques in the field of artificial intelligence to engineering design, the scope of the research work is multi-disciplinary in nature. The necessity for an in-product, contextual recommendation capability necessitates the consideration of disciplines such as machine learning, software development in addition to traditional engineering design. While the research work either utilizes capabilities offered by these disciplines out-of-the-box or results in evolutionary improvements to each some of these topics, the cumulative impact of the research work would be novel in nature. Thus, the research work is scoped to two particular applications, first to a design application that utilizes a Model-based Systems Engineering approach for the design of unmanned aerial vehicles with the aim of identifying patterns in observed data to improve performances of new generated designs and second to a commercial application, Siemens NX, that enables computer aided design where in an expert engineer's actions and decisions are identified and tailed in order to generate in-product and contextual recommendations for others.

1.9 Organization of the Dissertation

Having introduced the motivation and the goal of the research work, the following passages outline the organization of the dissertation. The document is organized into three sections. The first section reviews the state-of-the-art in the field of automation of design systems and decision support systems and establishes a methodology for the research carried out. These are covered in the CHAPTERS 2-6, with CHAPTER 2

reviewing the state-of-the-art in application of decision support systems to the engineering domain in the research and industrial setting. CHAPTER 2 concludes with a set of observations about the shortcomings of the established approaches. CHAPTER 3 proposes a means to address these shortcomings with the development of the primary hypothesis and the proposal of a generic computational framework. A research methodology is then established to aid the development effort of this framework. The chapter concludes with the identification of a set of three research areas and associated research questions that are to be addressed in order to develop the proposed framework. CHAPTER 4 addresses the three research areas identified and provide justifications the hypotheses developed for the research questions posed.

The second section of the dissertation documents the developed framework and its application to a set of three applications ranging from a canonical problem to its implementation to a commercial-off-the-shelf application. CHAPTER 5 documents the software application and the framework developed for the purpose of automation of design systems and also introduces the methods and algorithms considered in the implementation. CHAPTER 6 and CHAPTER 7 each address one instance of application of the framework for the automation of engineering design process (in CHAPTER 6) and the automation of an engineering design application (in CHAPTER 7).

CHAPTER 8 concludes the dissertation with a look back at the accomplishments of the thesis work and proposes a set of future developments to further extend the developed application.

CHAPTER 2. GAP ANALYSIS

Having introduced the background and the premise for the research work, it falls upon the author to now tackle the evaluation of the state-of-the-art in establishing potential areas for contribution, prior to establishing the methodology for the research work. The current chapter performs said evaluation of the state-of-the-art and is segmented into two. The first section of the chapter deals with the background of decision making and the processes involved in decision making in humans, while the latter section deals with solutions available for the implementation of the processes for decision making in a computational framework, and the evaluation of a set of the alternatives of these computational frameworks. The section concludes by highlighting some of the shortcomings of the practices in the development of these frameworks leading to a set of observations that guide the research methodology.

2.1 Decisions and Decision-making

2.1.1 Decisions

Decisions are central to all beings. Decisions are made every day by every one of us in one form or another. Decisions that have little impact on our lives are made every day quite frequently and, often, without much thought while those that are of greater significance are deliberated to greater length to judiciously arrive at, what is perceived to be, the best possible decision. The significance of a decision made and hence the problem of decision making, itself, has been identified to be influenced by three primary factors, the problem context, cognitive capabilities of the decision maker and the social implications as perceived by the decision maker, as illustrated in Figure 2.1. In fact, every

scenario that necessitates a decision, from the perspective of the decision maker, is unique owning to the impact the decision has on the decision maker. That is, it is the result of the influence of a decision on the cognitive capabilities of the decision maker that brings about the differences in the decision driving circumstances as no decision maker can return to a previous state of knowledge or the status-quo [22].

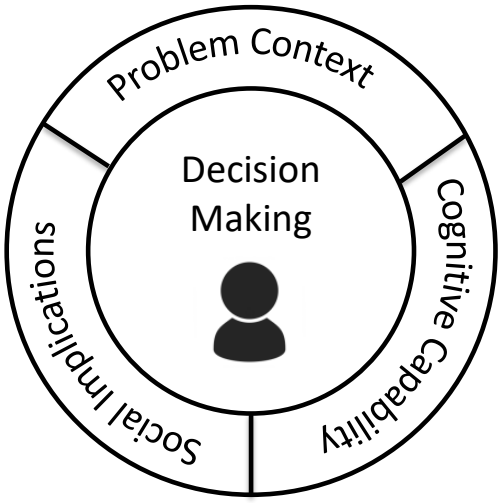


Figure 2.1: Factors affecting the decisions as identified by [60]

2.1.2 Types of Decisions

Decision behavior in humans, often, varies depending on the circumstances that drive the decisions being made to an extent that small changes to the circumstances could results in significant changes the final decision made. This is observed in the scenario where for the same problem context, given a different number of alternatives to choose from, the manner in which information is processed to arrive at a decision has been observed to be significantly different [60]. In general, it is a comparison of the available alternatives that drives the process of decision making. A common distinction in these

types of comparative decision making is as originally described for the purpose of management systems by the “strategy pyramid” [61] in which the decisions are classified as strategic, tactical and operational. But the nature of engineering design is such that, in addition to the three comparative decision types, there is usually a fourth. This fourth decision making type is one that relies on the ability to draw similarities between decision circumstances and is often terms the recognition-primed decision making [62]. Such decisions involve little or no comparison of alternatives but, instead, rely on instinct to dictate the decision made. It has to be noted that this structure of decision making in the strategy pyramid indicates that the decisions with higher consequence are those of the strategic nature while the instinctive decisions are ones that are made quite often and rather mundane in nature. Given that there is a recognizable pattern to the circumstance dictating the decision; it aligns well with the core concept of programming, where the patterns in decisions and their circumstances from the past can be exploited to arrive at decisions in new but similar circumstances. Further, the mundane nature of these decisions is the primary target for the task of automation, which attempts to remove the burden of making tedious and mundane decisions in order to free up the decision maker to more pressing and demanding tasks.

2.1.3 Topology of the Problem Context

A further classification of decisions comes from the circumstances driving the decisions made. A *cynefin* model [63] categorizes the problem contexts into four, illustrated in Figure 2.2 and described below,

- The *known* problem context where a complete description of the cause and effect of a circumstance is available to the decision maker in addition to the information

regarding the choices and their consequences. Decisions made under these circumstances are dictated by the recognition of patterns leading to actions based on experience, i.e., recognition-primed decisions.

- The *knowable* problem context where the cause and effect relationships are clearly defined, but there is insufficient information to accurately predict the consequence of the decision made. Decisions made in problems of these settings involve a more detailed analysis of the circumstance aiming to understand and develop models representing the trends in the decisions based on similar experienced situations. Thus, as with the known problem context, problems of this type are characterized by their repeatability.
- The *complex* problem context where a considerable number of interactions in the cause and effect of decisions that makes the prediction of the consequence of one single decision difficult. Circumstances classified as complex are characterized by an underlying uncertainty in their impacts which makes the process of decision making difficult. In such scenarios, decisions are often based on subjective judgements rather than objective facts.
- The *chaotic* problem context in which there exists no pattern dictating the cause and effect of a decision hence rendering it impossible for the determination of consequence of the decision. Decisions in such problems do not rely on any analysis but instead require an exploratory search of the available alternatives or require the simplification of the problem to one of the other contexts in order to utilize an analytical approach for the decision making.

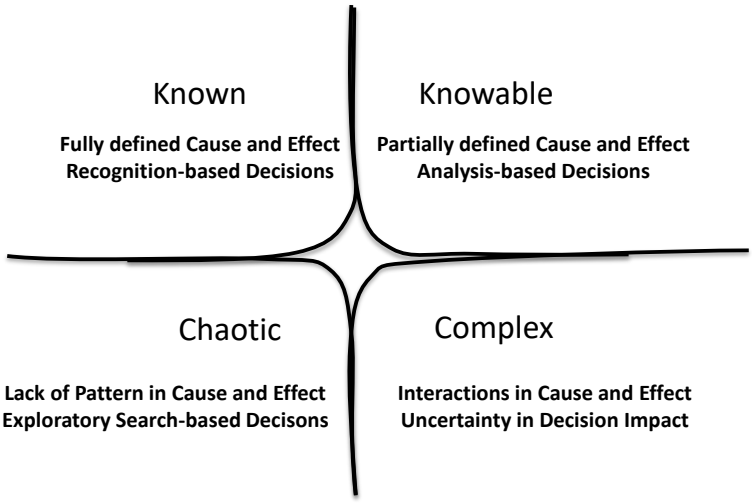


Figure 2.2: Categorization of the problem context based on Snowden's Cynefin model [63]

An alternate, yet synonymous, view of the topology of the problem context is attributed to Simon [64], [65] who classified the problems into programmed or structured and ill-structured or non-programmed. While originally classified as two alternatives of the problem context, the structured and unstructured problems can be viewed as two ends of a spectrum of problems that vary in the definition of the problem context. A structured problem would be one in which all the elements of that lead to the decision, such as the data, process and the analysis carried out, would be determinate and could be generated through a rigorous analysis of the problem. Such a problem enables the development of a framework that can be utilized to mimic the process of decision making and hence coincides with what is typically understood as being programmed. The unstructured problems, on the other hand, while having the same components as the structure problems, i.e., the data, process and analyses, are characterized by an absence of a definition of a methodology for the process of decision making. This lack of framework for decision making implies that each decision maker could utilize a different combination of the three attributes to arrive at a decision indicating an absence of

understanding in portions of the problem. Along the spectrum, in between the structured and the unstructured problems, would lie the semi-structured decision problems. These problems, while utilizing a standardized methodology in the decision making also rely on human judgement and experience to make the decision.

Table 2.1 [61] illustrates the orthogonality between the structure of the problem and type of decisions that are available while identifying representative problems that are applicable for each cell.

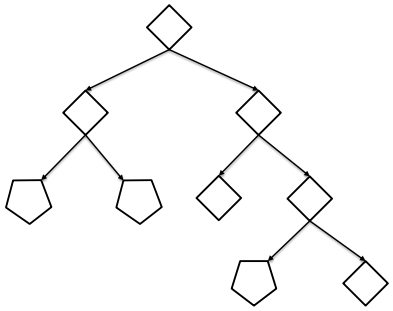
Table 2.1: Grid showning the orthogonality between the decision structure and the type of decisions executed [61]

<table>
<tr><th></th><th>Recognition-based</th><th>Operational</th><th>Management</th><th>Strategic</th></tr>
<tr><td>Structured</td><td rowspan="3">Contextual</td><td rowspan="3">Low-level Control</td><td>Mid-level Control</td><td>High-level Control</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>Unstructured</td><td>Unplanned Decision</td><td>Long-term Planning</td></tr>
</table>

2.1.4 Problem Representation and the Rational Decision Model

Having categorized the types of decisions and the circumstances that call for decisions, it now falls upon the author to introduce the means in which decisions are made. But prior to this analysis, means for the representation of the decision problem are introduced and the background for a commonly used decision model is addressed. Formal mathematical descriptions of the problem, while enabling a structured representation of

the decision problem, would not be the best means for communication of the decision-making problem. Thus, the following section reviews a set of representations for the decision problem that are suitable for a computational representation but also comprehensible to decision makers.



Condition Stub	Rules
	Condition Entries
Action Stub	Action Entries

Figure 2.3: Decision tree and decision table aproaches to decision modellling [66]

Perhaps, the simplest means of representing a decision problem and the decision alternatives available is represented by a *decision table*, illustrated in Figure 2.3. In this framework, it is the task of the decision maker to identify and relate a set of available choices, i.e., decisions, to the corresponding states of the environment. These states of the environment may be exogenous factors that represent the manner in which the environment reacts to a decision being made. The relationship between the states of the environment and the set of decisions are established through a matrix of consequences that dictate/predict the impact of making a certain decision for a given state of the system. While this modelling approach provides a comprehensive overview of all the choices and the consequence of those choices, the framework requires knowledge about the state in which the environment is at any point a decision is to be made. Further, the consequence of a decision need not necessarily be numeric in nature and could remain descriptive which does not lend itself to comparison due to the lack of objectivity, making the choice

amongst the decision alternatives difficult. The mathematical representation for the decision table follows from [22] as,

$$A_i \oplus \phi_j \xrightarrow{P,H} c_{ij}$$

where, A represents the decision made, ϕ represents the state of the environment, c represents the consequence perceived, P represents the problem faced by the decision maker, and H the history of events observed by the decision maker.

The two issues with regards to decision tables can be alleviated through the use of a specification for the belief of and the preference for both the state of the environment and the consequence as defined by the decision maker, respectively. While decisions made under such settings still remain subjective, this approach permits a comparative analysis of the possible actions. One of the most common means of defining these preferences is through the creation of a *subjective expected utility* (SEU) estimate which attempts to model the belief in the environment state as a distribution about the possible states that the environment can occupy, and the preference for the decision as an expected utility of the consequence of the action. It has to be noted that decision tables, through this extension, are applicable to structured and semi-structured problems which reside in the known, knowable or complex problem domains, and enable decisions of any of the four types defined in the modified strategy pyramid. This model for decision making with the utilization of SEU has been shown simulate the behaviour of a decision maker adhering to the axioms of perfect rationality [67], a characteristics that is strongly desired in the development of computational alternatives for decision makers. While such behaviour is certainly ideal, reality is often quite different as societal circumstances have a significant impact on the decision maker biasing the decisions away from the rational.

As an alternative to the decision table, a typically used representation for the decision problem is the *decision tree*, illustrated in Figure 2.3. A decision tree is a means of representing a sequence of decisions in which the set of possible decision alternatives from an initial state of the system are plotted against the progression of time. As with the decision table, decision trees require an enumeration of the decision alternatives and the resultant consequence of the decision, but only requires the specification of the initial state of the environment. These characteristics can be viewed as being both a pro and a con of the decision tree as follows,

- the decision maker is able to make a sequence of decisions using a single tree and only requires information about the initial state of the environment in order to make these sequence of decisions
- given a decision tree and the state of the environment, the choice of an alternative would require the decision maker to back-track decisions made to the initial state to evaluate the tree's applicability to the problem context

It has been argued [68] that, in the presence of numerical attributes for the decision criteria, decision trees can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of its axes. Due to this representation, it is often the case that decision makers prefer handling smaller decision trees as these are easier to comprehend.

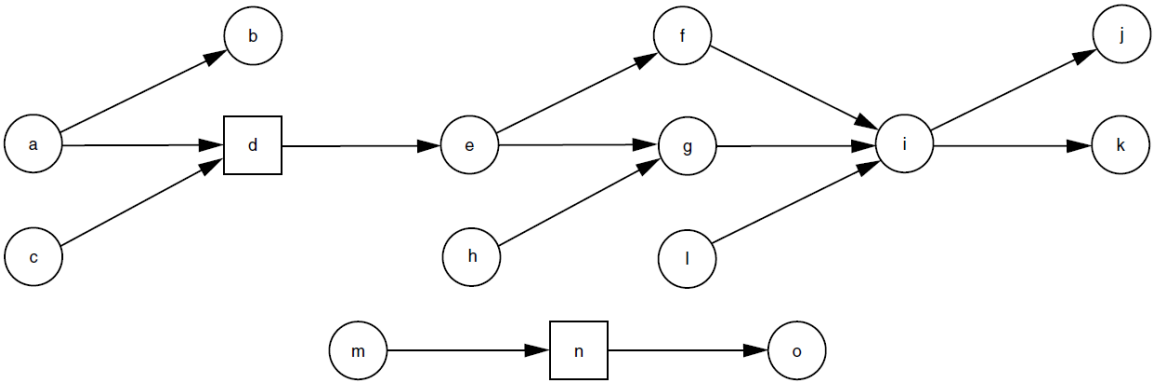


Figure 2.4: Generic Influence Diagram as defined by [69]

Another alternative to the representation of a problem is through the use of *influence diagrams* [69], illustrated in Figure 2.4. The influence diagram, in contrast to the decision tree is designed to represent the dependency between the decisions available to and the set of beliefs held by the decision maker, i.e., building a relationship between the decision parameters and the uncertain parameters. These diagrams require a specification of the set of beliefs regarding the states of the environment in addition to the specifications of the decision alternatives available to the decision maker from which a graphical model is developed where each one of the uncertain beliefs and the decision alternatives forms a node of the graph and the edges indicate the relationship between the decisions made and the "influenced" state. An intrinsic advantage of the influence diagram over the decision trees is its comprehensibility in the presence of large decision problems that typically results in them being favoured for real-world problems as applied for the problem of medical diagnostics [70]. In the example given in Figure 2.4 there is a presence of two decision nodes that relies on knowledge of a set of preceding nodes, in the first case that of a and c and in the second case that of m, while the remaining portion of the graph indicates the influence of the belief of the environment state on other state parameters.

It has to be noted that owning to the complexity of decision problems, it is often the case that decision makers rely on a complementary set of representations in order to model the context to guide the choice of the alternative. Thus, there is rarely a necessity to evaluate amongst the alternatives means of representation to trigger the decision analysis process [22].

2.1.5 Decision-making Process and Decision Analysis

While there exist numerous models that have attempted to explain the process of decision making [71], most rely on Simon's model [72], illustrated in Figure 2.5, as the basis for the development of a rational decision-making process. Simon's model for rational decision making is divided into a set of three primary phases followed by an implementation and analysis phase. Simon argues that the decision-making process begins with the intelligence phase in which the problem being addressed or the opportunity for an improvement in some established methods is identified. The phase entails the identification of the three characteristics of the decision-making process, i.e., the collection or generation of all relevant data to inform the modelling of the decision problem, the identification of the process for the evaluation of the decision problem that results in the selection of a decision-making technique and also the definition of the evaluation metrics that serves as the compare various decision alternatives. These parameters are based on the requirements and constraints posed by the decision problem.

The intelligence phase is followed by the design phase where the primary focus of the decision maker is the identification of decision alternatives. In the absence of a suitable decision alternative, a detailed investigation of the decision problem has to be performed to generate a suitable set of alternatives. Having generated a suitable set of

decision alternatives, the identified analysis is to be exercised in order to generate the set of metrics to enable a comparison amongst the alternatives, concluding the design phase.

The final phase as identified by Simon in his model for rational decision making is the choice phase. The choice phase involves the comparison of the set of identified alternatives with the goal of selecting the "best". The definition of the best alternative is, of course, subject to the preferences of the decision maker thereby making biases inherent to the process of decision-making. The choice exercises the decision-making process identified in the intelligence phase so as to filter out subpar alternatives yielding the best. Owning to the nature of the decision-making processes, parameters associated with the methods are defined thereby enabling the comparison of the alternatives in a quantitative sense. The definition of parameters of the decision-making parameter, too, permits the infusion of bias into the final decision. The phase then concludes with the selection of an alternative.

While not originally included by Simon, research later have included the implementation of the alternative as an intrinsic component to the decision-making process. The phase involves the realization of the decision, where an *action* is taken committing a finite amount of resources with the hope of realizing the expected consequence. This effect of the realized action is analysed to ascertain the degree to which the realized product meets the expected consequences, thereby triggering a diagnostic on the failure to meet the expected consequences with the goal of identifying the source of the difference, triggering an iterative decision-making process.

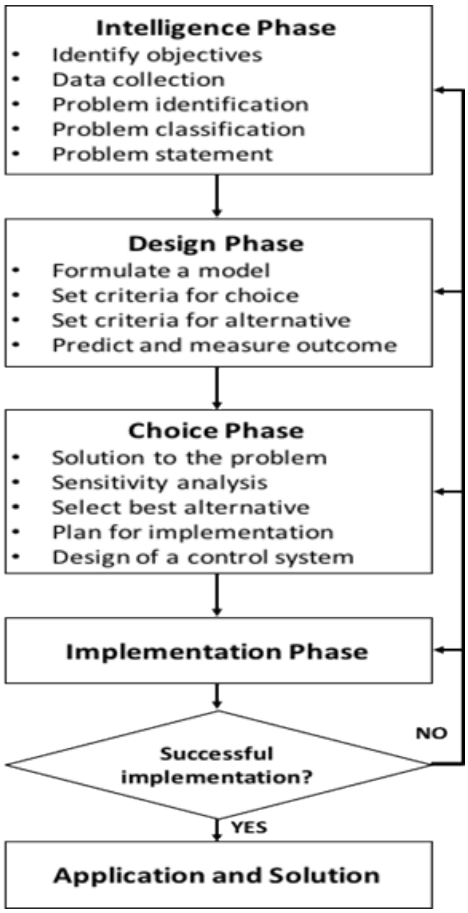


Figure 2.5: Simon's model for Rational Decision-making process [64], [73]

At this point, it is essential to distinguish between an action and a decision. A comprehensive description of the differences between an action and a decision as applied to a generic engineering context is provided by Hazelrigg [74] and is summarized here. Hazelrigg defines a decision as a commitment to an action of uncertain effect involving some irrevocable allocation of resources, which is made in the present while being different from the actual action as it is contained in the mind of the decision-maker and involves a choice from a set of alternatives with the aim of reaching a certain desired pre-determined goal. Further, Hazelrigg defines actions as the physical manifestation of a decision on the environment.

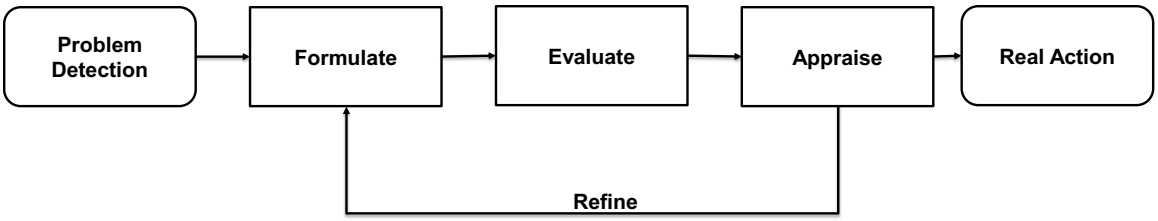


Figure 2.6: Howard's model for Decision Analysis [75]

Howard [75] proposed a framework that attempts to explain the manner in which the decision analysis is undertaken in humans. The framework, illustrated in Figure 2.6, attempts to reason the manner in which an analysis of the decisions made by humans are carried out so as to impact and alter the nature of future decisions. The framework of decision analysis, as defined by Howard, functions to "systematically transform decision problems that are difficult to solve and understand into ones that are clear and lucid by a sequence of transparent steps". Howards process of decision analysis comprised of three primary steps working towards the goal of providing insight into a decision problem so as to result in the recommendation of an action. This process of decision analysis is triggered with the *formulation* of a model for the opaque decision problem. Howard termed this representation a "decision basis" and proposed that the basis be comprised of three entities that help the decision-maker arrive at their decisions, i.e., the alternatives from which to choose, the information relevant to the problem and biases or preferences of the decision-maker. The basis when computationally *evaluated* through a series of analyses yields the alternative that would be most logically consistent with the defined parameters. This is followed with an *appraisal* of the analysis to not only determine the logical consistency of the recommended decision, but also if it is persuasive enough to be preferred by the decision-maker. If such an appraisal process indicates any shortcomings in the analysis process, the formulation is to be refined to be a more accurate

representation of the decision problem. The refinement may be repeated until the appraisal process identifies a decision alternative the is appropriate for the decision-maker, i.e., one that is both the logical and preferred by the decision-maker.

Having identified a framework for both the rational decision making and decision analysis, it is essential to note that the decisions made by humans are driven by biases and preferences leading to the two models of decision-making, the rational (or normative) and the real (or descriptive) decision-making model. Given the rigid framework governing the normative models of decision-making, there have been considerable developments in computational systems that aim at supporting the decision-making process. These computational systems are often termed Computer-based Information Systems, or CBIS in short and the topics of interest, decision-support systems and expert system form a subset of such computational systems.

2.2 Decision-Support Systems

With developments in the field of artificial intelligence, there have been attempts to automate the process of decision making with the development of *decision-support systems* – computational programs attempt to replicate human like decision making based on the knowledge stored to them. In the field of engineering design, these systems are often termed *expert systems* as they play the role of an expert engineer by assisting the process of decision making. The systems typically have vast amounts of domain knowledge applicable to the problem's contexts stored in them making the replications of expert-like decisions possible. Such systems are designed so as to accept knowledge from "expert engineers" and develop an encoding mechanism in a means so as to not only enable the application of the encoded knowledge to other problems in the manner that an

expert engineer would, but also enable reasoning as to the decisions that are made by the system [76]. The analysis of such computational systems begins with a dissection of the decision support systems, starting with a technical definition [77] of one such system,

---

Decision Support Systems

Decisions support systems are anthropocentric and evolving information system that are meant to implement the function of a human support system in order to overcome the limitations of a human decision-maker while solving complex and complicated decision problems.

An analysis of the definition highlights three characteristics of the decision support systems,

- Decision support systems are evolving informational systems. While insight into the evolving nature of the system is not attained by the above definition, a decomposition, performed later, of the system provides this answer. Informational systems, in this context, refers to computational software that relies on the utilization and communication of information in the process of making decisions although the type of information utilized differs based on the type of decision support system developed.
- The primary directive of decision support systems is the support human decision making in circumstances where human decision-maker capabilities are limited, for example, due to lack of experience or lack of information. While the definition

does not limit the application of decision support systems to a particular type of problem, analysis [78] has demonstrated that such systems are predominantly used to address semi-structured problems and tackling tactical and strategic decisions associated with the problem. This is illustrated in Figure 2.7 where the Raymond's interpretation of decision support systems is overlaid on the Gorry and Morton [61] grid.

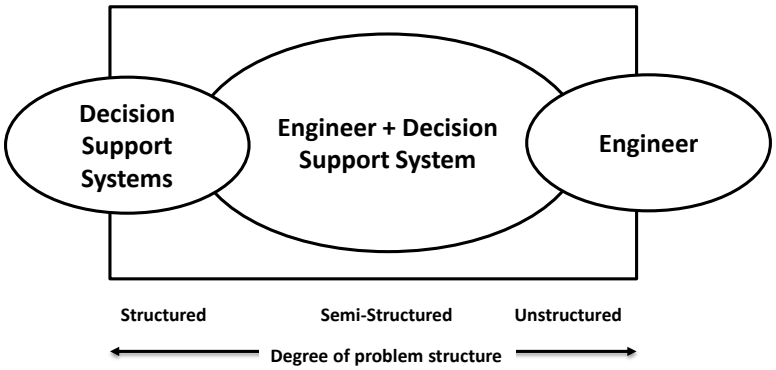


Figure 2.7: Raymond's interpretation of the Gorry and Morton grid indicating the type of problems suited for decision support systems [78]

- Finally, the definition highlights that decision support systems are designed to operate on a particular problem context, i.e., these computational systems are designed to operate in a certain area and are configured to operate under strict guidelines in order to arrive at decisions.

Having gained an understanding of what decision support systems are, the different types of decision support systems have to be identified. To accomplish this, Power's [79] taxonomy for the classification of decision support systems is utilized. The classification is driven by the identification of the primary component that drives the decision making leading to five distinct classifications that are described briefly below,

- Model-based systems in which a qualitative model drives the process of decision making in which the parameters for the model are provided by the decision maker,
- Data-based systems where the decisions are based on analysis of vast quantities of data,
- Knowledge-based systems that recommend or suggest actions to a user based on specialized information stored in a knowledge bank.
- Communication-based systems whose primary function is to enable decision-relevant collaboration based on communication technologies.
- Document-based systems that rely on document analysis to assist the decision-making process, for example search engines [80].

As indicated, decision support systems are directed at strategic and tactical decisions, a classification that perhaps is more managerial rather than engineering. But the nature of two of the identified systems, the knowledge-based and data-based decision support systems are such that they can be extended to operational and recognition-based decision making. In the engineering field, such systems are called knowledge-based information systems or expert systems [81], the primary topic of this dissertation.

2.3 Expert Systems

It is the nature of human beings to rely on past experience to derive decision rules to address problems requiring the evaluation of decision alternative to meet a set of requirements [82]. The field of engineering design is one where there is often a choice among several alternatives at design decision. This nature of engineering design has led researchers [59] to characterize the engineering design problem as being similar to that of

sequential decision making, i.e., essentially one involving the arrival of a series of decisions based on the nature of the design. But relevant experience, in such scenarios, does not come by easily. It has been identified that experience in the field of engineering design typically takes several years [76] to attain and this is, typically, not transferrable across domains or engineers. In such scenarios, often as an alternative to expensive and time-consuming workforce training, expert systems are utilized. Such systems are prevalent in several fields such as engineering decision making [73], medicine[35], finance [83] and computer-aided design [84]. To better understand expert systems, it is essential to first define the term.

Expert Systems

Expert Systems are intelligent computer systems that are comprised of heuristic rules and detailed domain facts and use knowledge and inference procedures to solve difficult problems that require significant human expertise for their solution.

The definition of expert systems given above [85] establishes that expert systems form a subset of the decision support systems, in particular knowledge-based decision support systems. Building off of this, one could identify a set of characteristics of expert systems that are highlighted in its definition, given below:

- Firstly, the expert systems are computational software that are developed to address a certain problem. Hence, one such system would rely on software

development practices in its development and further, would only applicable to the problem context defined during its development.

- These systems rely on heuristic rules and domain facts in the process of making decisions. These heuristics and domain facts need to be provided by expert engineers and represented in the computational software so that they can be called upon at any time.
- Expert systems rely on an efficient means of encoding knowledge gathered through years of intensive exercise on relevant problems by a human expert into a knowledge database, such that the developed encoding mechanism enables recovery of decision rules leading to an action.
- Finally, expert systems need to rely on a decision-making scheme that can utilize recovered encoding of the knowledge in order to evaluate several decision alternatives and identify the "best" alternative under the given problem context.

In summary, given the two paths in which a human decision-maker can arrive at a decision, i.e., the path where human reasoning and knowledge is used to make a judicious choice of a decision and the utilization of an expert system to recommend a decision, the primary purpose of an expert system can be viewed as ensuring a compatibility between the two generated decision alternatives. Hence, given a problem context, there should, at all times, be an agreement in the decision made by an expert engineer and the decision recommended by the expert system.

Taking a brief digression from expert systems to categorize the field of artificial intelligence, one would find that on one end of the spectrum of capabilities involving artificial intelligence are the methods and capabilities that are enabled through machine

learning techniques while on the other end of the spectrum would be the decision support and expert systems. This categorization is illustrated in Figure 2.8. While these two branches (or categories) do not, traditionally, overlap, the goal of the current thesis work is to bridge the gap between the two branches thereby leveraging methods offered by the framework of machine learning in the development of expert systems for its application to engineering design problems.

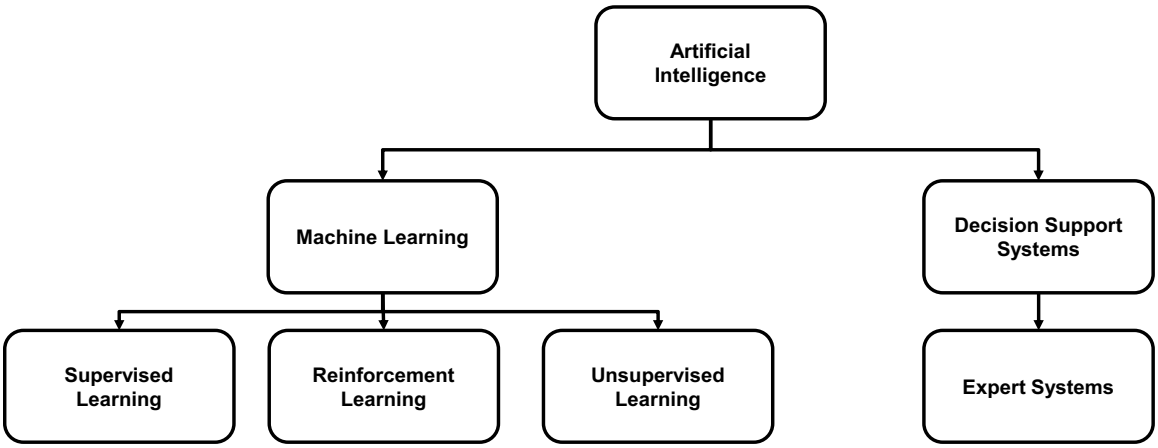


Figure 2.8: Decomposition and categorization of the artificial intelligence domain

2.3.1 Expert Engineers and Expertise

In order to better understand the role and behaviour of an expert system, it is essential to first gain an understanding into the term *expert* and its derivative *expertise*[2]. While historically expertise in a certain field has been treated as an innate ability of an individual, recent definitions of the term have instead indicated that expertise to the skill and knowledge gained by a practicing individual in a certain field over a course of several years [76], [86], [87]. That is, expertise is something that can be gained by any practicing engineer regardless of their "natural" abilities. The statement is backed by

[2] In the ensuing discussion an engineering setting is considered, and the term *expert* is often replaced with *expert engineer*.

research work demonstrating the progression of an individual's the performance in a certain task with the amount of time spent on the problem [87] and is illustrated in Figure 2.9. Further, since expertise in a certain field can be measured as the performance of an individual in the field, one could use the performance measure as a surrogate for expertise, thus relating expertise to training or exposure duration.

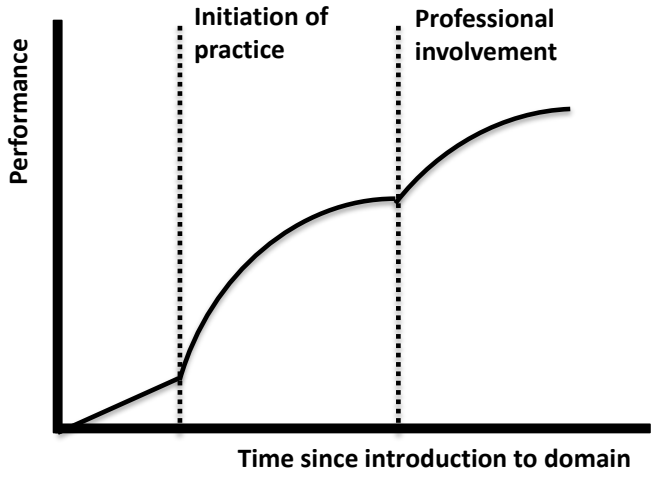


Figure 2.9: Phases of development towards expertise [86], [88]

Dr. Benjamin Bloom [88] proposed a three-stage development approach in which he argued expertise can be attained in a certain field. Drawing parallels to the phases proposed by Bloom [88] to the engineering domain, the first of these three stages involves the introduction of a practitioner to the field of interest. Such a phase either starts with the practitioner being exposed to the engineering field as a student in a formal academic setting or through deliberate investigations by the practitioner to gain some knowledge about the field. In both settings, the practitioner's performance is often poor, and is guided by educational guides such as in an academic education or other resources. The phase serves to pique the practitioner's interest in the field, which if successful triggers the second phase of learning. The second phase would correspond to the rigorous

education (or training) and the acquisition of professional practice in the engineering field where the practitioner would adopt a full-time commitment to improving his/her performance in the field. The culmination of such a phase would be the transition of the practitioner from an academic setting to a full-time professional career in the engineering field, which would correspond to the third phase of performance development. In the engineering domain, the third phase would involve the practitioner being exposed to problems that customized to a small section of the engineering field, thereby focusing on honing a select aspect of the practitioner's skills. At this point, the practitioner would have likely made a full-time commitment to the profession such that they are able to live off of it. Thus, "*the best human expertise is often a result of years, perhaps decades, of practical experience"* [76]. To identify a concrete measure of the duration it takes to attain expertise in the field of engineering and science, it has been observed that it typically take on the order of a decade or longer for an engineer to attain expertise in the domain of practice [87], [89].

2.3.2 Desired characteristics of an Expert System

Given the long durations in acquisition of expertise and expertise not being readily transferrable across engineers, expert systems have traditionally been used as alternatives. As with most computational programs, expert systems, too, are characterized by speed, accuracy, reliability and cost-effectiveness, to an extent that they often outperform the human equivalent. But in order to justify the use of expert systems as a rational replacement for an expert engineer, certain key, and often overlooked, characteristics of the human equivalent have to be computationally reproduced. These include the ability to justify, reason and explain a decision, the ability to make decisions in the presence of

missing information, the ability to translate knowledge, and the ability to acquire knowledge through experience. These characteristics establish a set of criteria that can be used to subjectively evaluate any expert system and compare them to the performance of an ideal expert in the field. Table 2.2 highlights these characteristics and provides a description of the interpretation of these evaluation criteria.

Table 2.2: Evaluation criteria guiding the comparison of state-of-the-art Expoert Systems

Desired Characteristics	Interpretation
Adaptability	A measure of the amount of modifications required to apply the system to a problem in a different domain.
Learning Capability	A measure of the system's capability to adapt to new observations and self-correct based on exploration.
Inference Engine Development	A measure of the difficulty in training and tuning of the inference engine.
Explanation Capability	A measure of the system's capability to explain a recommended decision.
Uncertainty Tolerance	A measure of the system's capability to handle absence of data or uncertainty in decisions.

Maintainability	A measure of the ease with which a continuous development cycle can be incorporated into the system.

Table 2.2 (Continued)

2.3.3 Constituent components of an Expert System

The traditional structure of an expert system with the constituent interactions and relations is illustrated in Figure 2.10. It indicates that expert system are comprised of three primary components [22], [76], [90], the knowledge-base, that stores the information supplied by the expert engineer, an inference engine, that utilizes decision-making rules to evaluate decision alternatives, and the user interface, that eases the utilization of the system and serves to both accept information and display decisions, reasoning and the associated logic.

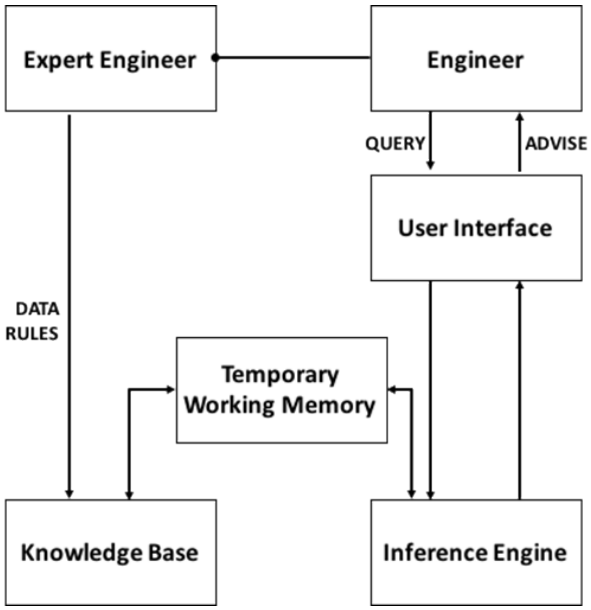


Figure 2.10: A representation of the components of an Expert System and their interactions [90]

- The knowledge base stores all the necessary information related to the heuristics that assists problem solving process in the application domain. The knowledge base typically contains the theoretical concepts associated with the problem domain, the empirical rules obtained from an experienced expert, models that are a representation of a collection of models and high-level strategies that dictate the decision-making behavior when a problem is encountered. Efficient encoding and representation mechanisms are typically adopted enabling quick access and retrieval of knowledge to aid the process of decision making.
- The inference engine serves as the computational block emulating the decision-maker. The inference engine attempts to break down a problem in order to search the knowledge base for decision-making strategies or relational models and thereby draw inferences based on the results found. This process utilizes the theoretical knowledge stored and practical observations made by expert engineers. The inference engine interacts with a temporary local memory unit, the working memory, where the problem specifics and information regarding the solutions identified are stored.
- The user interface serves as the visible interface that an engineer interacts with. The user interface provides the results and any additional information sought from the inference engine.

The operation life cycle of an expert system can be viewed as being comprised of two phases. The first phase would involve an expert engineer interacting with the system to define the domain knowledge that is to be encoded within the knowledge base. This establishes the basis used in the process of decision making. In such a mode of operation, also known as the knowledge acquisition mode, the user interface serves the function of eliciting information from expert engineers. The information gathered is utilized by

knowledge engineers who develop appropriate encoding and representation mechanisms such that vast quantities of knowledge can be easily stored, parsed, analyzed and retrieved from within the knowledge database at any instant of time.

The second phase of operation involves a design engineer, or a novice engineer who interacts with the system aiming to acquire guidance in the process of decision making. In such a scenario, the user interface would relay information about the active problem context to a working cache memory storing all the information about the problem that is necessary to facilitate decision making. The inference engine, in the process of decision making, queries the cache for information about the problem. The nature of the query depends on the inference mechanism used. Commercially used interference mechanisms include forward and backward rule-chaining [76], [90] and tree searches [73]. Rule-chaining approaches rely on arriving at conclusions by chaining at set of conditions to arrive at a conclusion in either direction, while the tree search methods rely on comparison of nodes in the knowledge base represented as a tree to the problem context to identify the closest matching leaf node. Once a decision is arrived at, the user interface displays the conclusions, reasonings and justifications for the decisions to the user. It then falls upon the user to implement the recommended conclusion, and the process repeats until the entire sequence of decisions have been exhausted.

It is essential to note that while the separation of the operation of the expert system into two phases provides a clear distinction between the roles in which the user functions, these phases need not be sequential. While a populated knowledge base is bare minimum necessity to enable the transition to the second phase of operation, dynamic changes to the knowledge base can be permitted by implementing a decoupled offline knowledge acquisition process. In such a setting, the knowledge base continuously evolves with

incorporation of new information and the inference mechanism would adapt to this evolving knowledge base enabling a growing and dynamic system. Of course, in order to enable such a capability, appropriate measure have to be taken during the architecting of the expert system.

2.3.4 Types of Expert Systems

The classification of expert systems is predominantly driven by the inference mechanism used. Alternate classifications exist on the means in which the knowledge is represented and encoded. The following passages introduce some of the commonly used expert systems that have found commercial success.

2.3.4.1 Rule-based Expert Systems

Although the knowledge processing capability of human beings is too complex to be represented in a computational routine, with expertise humans gain an ability to express knowledge required for problem solving in the form of decision rules [91]. These rules guide the decision by establishing a conditional relationship between a conclusion (or consequent) and an antecedent (or condition) through the use of IF-THEN statements. Traditionally labelled as categorical knowledge, the IF-THEN rule only contain logical relationships between facts, without any ambiguity [92]. Expert systems that mimic this nature of decision making rely on the representation of the domain knowledge in the form of decision rules and are termed rule-based expert systems. Rules-based expert systems are the most commonly used type of expert systems and have found applications in several fields such as engineering, medicine, mining, power systems, etc. The structure of a rule-based expert system is illustrated in Figure 2.11 where encoding of knowledge in

the knowledge based in the form is IF-THEN rules specifying a "relation, recommendation, directive, strategy, or heuristic", is shown.

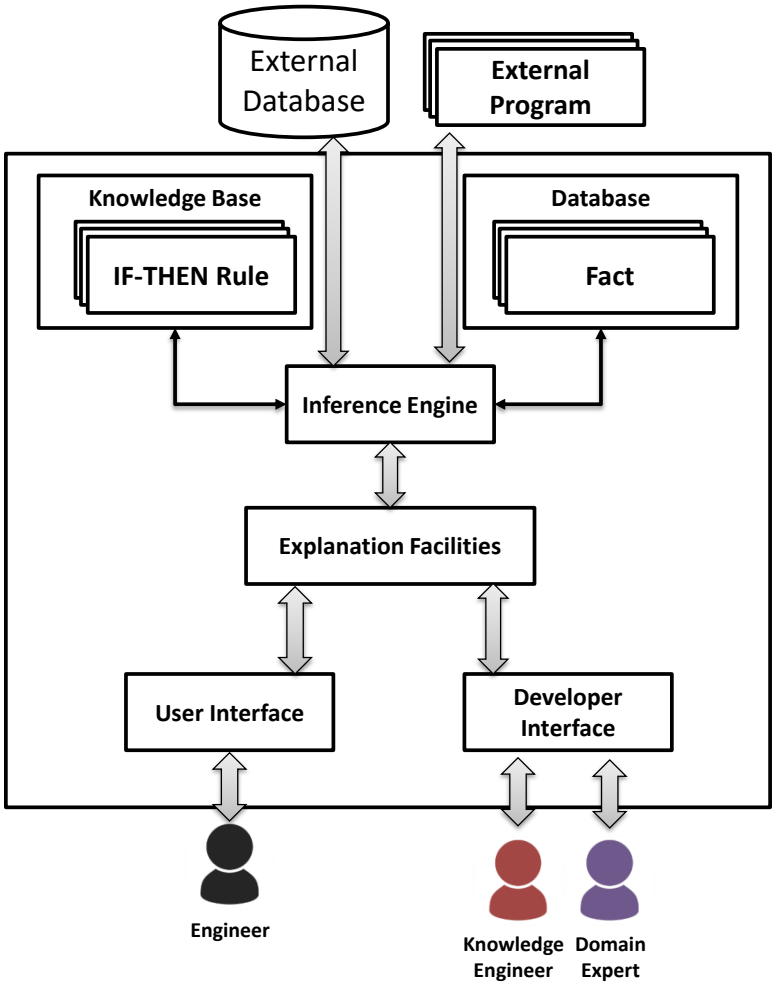


Figure 2.11: Structure of a rule-based Expert System [91]

In its simplest form, the structure of this knowledge representation is given as,

IF <antecedent>

THEN <consequent>

As with logic circuits, rule antecedents can be combined to build rather complex representations through the use of conjunction (AND) and/or disjunction (OR)

conditions. The antecedent of the rule is comprised of three parts, a relational metric, an operator and an associated value. The goal of such a representation is the generation of a mathematical expression enabling the comparison of the metric against the specified value such that the validity of the expression can be evaluated. The computational representation of the rule in the knowledge base involves the storage of the three term tuple representing the rule antecedent.

<antecedent>: (relational metric, operator, value)

On the other hand, the consequent can be of a complex nature where it is comprised of multiple clauses. In such cases, when the rule is satisfied each one of the clauses are triggered, with the details of the execution synchronicity depending upon the individual problem case considered and the architecting of the system. As with the antecedent, the consequent, too, is comprised of three parts, the relational metric (or a linguistic object), an operator and the value (or expression). In most cases, though, the operator is one of assignment, the linguistic object is preferred in place of the relational metric and an expression is used in place of a single value. In contrast to the antecedent with executes a logical comparison, the consequent executes the in-memory assignment when the logical conditions defined by the antecedents are met. But owning to the similarity in representation, the knowledge base treats stores the consequent in a manner similar to that of the antecedent of the rule.

<consequent>: (linguistic object, assignment operator, expression)

Owning to the nature of the definition of rules, it is relatively easy to define such rule-based systems where the representation of domain knowledge utilizes declarative or high-level language, such as Prolog [93], Lisp [31], etc., thereby making it easier for

expert engineers to define the knowledge in natural language (or close to it) rather than a complex programming language.

An inherent characteristic of rule-based expert systems is its ability to encounter conflicts as a result conflicting rule definition due to improper knowledge acquisition. Rule-based expert systems have the capability to resolve these conflicts where the mechanism utilized for conflict resolution depends on the inference mechanism used. In forward chaining inference, the conflicting rules are visited in sequence in a synchronous execution system, and in parallel in an asynchronous one. Thus, the asynchronous system would lead to inconsistency and would not be a viable setup. Several approaches have been proposed to alleviate this issue [81], [83], [91], [94]:

- A goal can be established for each consequent such that when the value or state associated with the linguistic object representing the goal is altered, the processing of the rule is terminated.
- Rules can be associated with priorities such that in conflicting cases, consequents of rules with the highest priority are executed.
- In the case of conflicting rules with different number of antecedents, the most specific rule is prioritized and executed.
- The time sequence of the rules is exploited such that in conflicting cases, rules that have been entered into the knowledge base more recently are prioritized and executed.
- Metaknowledge can be utilized to guide the expert system in the process of decision making providing the system a guideline about the encoded

knowledge. These "metaknowledge manifest as metarules that dictate the strategies for use of task-specific rules in an expert system"

An extension to the traditional framework of definition of explicit rules through the use of Bayesian reasoning [95] or *Certainty Factor* [96] permits the handling of uncertainty in the definition of rules. Uncertainty could arise from a variety of sources, such as, missing data, conflicting views of different experts, imprecise language in the definition of rules and weak associations in the definition of rules. In such scenarios, the rules structure can be modified to include a probabilistic definition such that in the case of Bayesian reasoning,

IF <antecedent>

THEN <consequent> [with probability P]

and in the case of Certainty Factor modeling,

IF <antecedent>

THEN <consequent> $[cf]$

where cf would represent the belief that the associated consequent would occur given that the antecedent is satisfied. Thus, in both cases, probabilistic modeling can be leveraged in expert systems to account for any uncertainty in the definition of knowledge rules. The Bayesian approach to modeling of uncertainties is supported by a strong mathematical background relying on probability theory. An inherent drawback of this approach is its reliance on statistical data in the modeling approach and also its

assumption of independence of the rule antecedents. On the other hand, certainty factors, being subjective estimates representing the expert's beliefs, lack the mathematical foundation of its counterpart and may be better suited in situations where probability measures are unobtainable.

Though rule-based expert systems offer a variety of advantages ranging from the use of natural (or high-level) knowledge representation, the decoupling of the knowledge representation from the subsequent processing steps to the ability to deal with incomplete and uncertain knowledge, they suffer from a complicated interactions between rules in large scale systems that makes the tracing of decisions difficult, the inability to learn and adapt the knowledge base and inference mechanism to new scenarios and also the inability to adequately index the knowledge base to efficiently search for the appropriate rules set that are to be evaluated typically leading to exhaustive searches.

2.3.4.2 Fuzzy Expert Systems

The definition of a rule in an expert system requires the specification of a real valued number in order to enable comparison during the inference process. Though rule-based expert systems establish a guideline for the replicating human like decision making, there is a tendency in experts to be vague in the definition of these comparison values. While other experts may have no issues interpreting these vague definitions, it is not possible for a computational system to do so. In order to address the issue of vagueness and ambiguity in the definition of rules, the concept of *Fuzzy Sets* [97] is borrowed upon. Fuzzy set theory enables the comparison of vague definition of objects and is based on the philosophy that all things are comparable and can be represented on a sliding scale. Relying on the theory of *Fuzzy Logic* [98] which restrains from making a

binary distinction in comparisons, but instead represents the truth of a statement as a real number in the range of 0 (false) to 1 (true), thus leading to multi-valued comparisons. Under the fuzzy set theory, each element belongs to a fuzzy set with a certain degree of membership, as a result of which under the multi-valued comparison rule, an evaluation of the element can be partly true to a certain degree. Thus, this real-valued representation would enable a graceful transition across the binary boundary that traditional comparison would resort to. Fuzzy logic finds application in a wide variety of products, such as household products, and control systems [99].

The relationship between the binary boundary and a fuzzy boundary is illustrated in Figure 2.12 where the horizontal axis represents the universe of possible values associated with the element (or variable) called the "*universe of discourse*" and the vertical axis would represent the membership value for certain set. Thus, the curves shown in the figure establishes a mapping between the variable value and the memberships to a certain set.

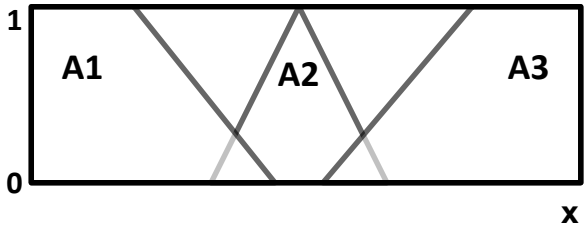


Figure 2.12: Membership functions for multi-valued relationships

The mathematical representation of the Figure 2.12 defining the mapping between an element of a set to its membership value is given as,

$$\mu_A(x) : X \rightarrow [0, 1]$$

where, μ_A represents the membership function, A represents the fuzzy set, x the element of a universe X. Having obtained a mathematical representation, its equivalent computation representation is tackled. The representation of a rule with the incorporation of fuzziness beings with the definition of a membership function. Typically used functions include linear, sigmoid, gaussian or pi functions, but are these are used to establish the degree of membership, computation time is an important consideration that has to be considered. Owning to this, most commercial applications restrict the modeling of fuzziness to linear fit functions [91]. This information is then encoded into a vector stored in the knowledge base where the information contained in case of the linear function would be represented as,

<Set>: (0/Value@0, x_1/Value@x_1, …, x_i/Value@x_i, …, x_n/Value@x_n 1/Value@1)

Using the above representation of a fuzzy set, *fuzzy rules* [97] can be defined in a manner similar to that of crisp rule of the rule-based expert systems, while retaining the fuzziness in the specification of the linguistic variable's values. This capability has been seen to have the capability to merge rules together effectively leading to a 90% reduction in the number of rules [91]. The process of inference using fuzzy rules varies slightly from that of the rule-based expert systems. In the classical setting, when an antecedent is evaluated as being true, all the corresponding consequents are assumed to be true as well. But in the fuzzy setting, since there is a degree of membership associated with the antecedent, their corresponding consequent would be partially true to the same degree. Thus, in these settings, in the presence of multiple rules, each rule contributes to a certain

degree to a consequent where these effects are aggregated to result in a single crisp value for the consequent. Having obtained a membership metric for the consequent, a process of *defuzzification* is undertaken to yield a crisp output representing the decision. Figure 2.13 illustrates the steps undertaken in the fuzzy expert system by which a decision is made. The structure of the system is identical to that of the rule-based expert system but for the incorporation of fuzziness in the definition of the rules.

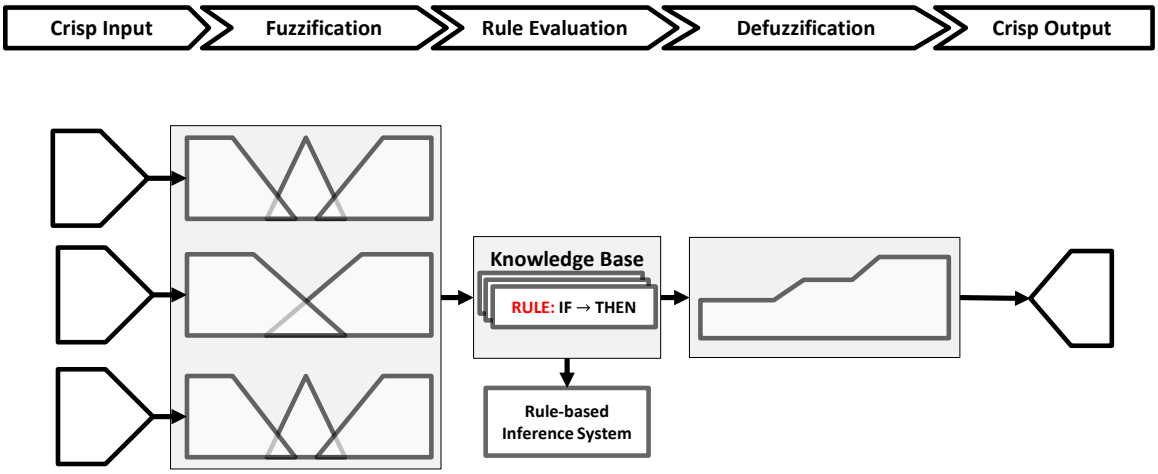


Figure 2.13: Steps involved in the decision-making process for a fuzzy expert system

Ignoring the obvious differences in the knowledge acquisition, the two keys steps that are different from the operational life cycle of rule-based expert systems are the aggregation of evaluated rules, the mathematical foundation for which is provided by fuzzy set theory, and the defuzzification of the aggregated metric. There are two types of fuzzy expert systems that are commonly used that differ in the method of aggregation used. These are:

1. **Mamdani-style inference system** [100]: Given a set of crisp inputs for each of the variable, the degree of membership is estimated based on the defined fuzzy functions. This is the process of fuzzification. With the degree estimates from the

membership functions, the antecedents of the fuzzy rules are evaluated such that a single metric is derived for each rule based on fuzzy set theory. These metrics are then mapped to the rule member functions that determine the degree of membership of the set of consequents, which too would typically be fuzzy in nature. These consequent memberships are aggregated through a set-based union resulting in a single fuzzy set for the consequent. This is followed by the defuzzification which results in a single metric representing the decision. While several methods exist for the defuzzification process [101], the most commonly used on is the centroid method which identifies the decision as being a the centroid of the aggregated set area and the consequent value as being the crisp value corresponding to the evaluated centroid.

2. **Sugeno-style inference system** [102]: The construction of an aggregate fuzzy set as an 2D surface results in a computational burden during the defuzzification stage when its centroid is to be evaluated. To alleviate this issue, the membership functions for the rule consequents can be replaced with spike functions thereby enabling the development of a zeroth-order approach. The spike function effectively leads to the final decision being a weighted average of all the rules consequents.

While fuzzy expert system shares most of the advantages of rule-based expert systems, they are able to account for ambiguity in the definition of the rules, a capability that broadens the scope of applications for the field of expert systems. But the necessity for the definition of fuzzy sets exacerbates the reliance on expert engineers thereby worsening the knowledge acquisition process. The calibration of the fuzzy sets and rules

is found to be the tedious and laborious aspect of the development process of a fuzzy expert system [91].

2.3.4.3 Frame-based Expert Systems

Frame-based expert systems differ from the previously introduced types as these systems utilize a different means for the representation of knowledge. The frame [103] forms the basic entity that is used to represent knowledge in such systems. Frames are defined as "a data-structure for representing a stereotyped situation". These data-structures are used to the knowledge representation method of choice for the frame-based expert systems where each entry in the knowledge base represents a particular object or concept. Frames are identified by their names and are comprised of a set of attributes or slots. Each attribute is assigned a value or a procedure that can be called upon to extract a value. It has been argued [103] that frames are the most natural way to represent the knowledge and is well aligned with the means in which humans organize perceive the surroundings and organize knowledge. From a computation perspective, a frame-based system represents an application object-oriented programming to expert systems. Due to the presence of all the necessary attributes for the representation of a concept, frames have been noted as being an efficient and concise means of representation of knowledge. Extensions to the attributes-value data structure can be achieved by the use of facets. Facets enable the association of additional properties with the attributes defined on a frame that can help the knowledge engineer in the representation of the knowledge. While attributes act as containers for knowledge, frames also support the capability to manipulate knowledge elements. These are achieved through the use of methods and demons, both associated with frame attributes. These capabilities enable frames to handle

both procedural and declarative knowledge, a capability that rule-based systems lacked. Methods represent the procedural code that is executed on demand while utilizing on knowledge stored in the frame. Demons on the other hand manifest in the form of IF-THEN rules and represent the declarative aspect of the frame.

As with object-oriented programming, each instance of the frame occupies a unique location in memory and can be related to other frames through a set of three relational possibilities: inheritance (generalization), aggregation (composition) and association. The concept of inheritance leads to the presence of classes and instances that enables the development of an ontology to guide the representation of knowledge. Though the creation of efficient and generic structure for the representation of the knowledge may be a tedious and difficult task for the knowledge engineer, it can greatly simplify the work of an expert engineer during the knowledge acquisition phase. This is due to the nature of inheritance where characteristic and attributes of a class are assumed by the inheriting instance thereby reducing the amount of knowledge that is supplied by the expert engineer. The capability of inheritance leads to the representation of knowledge in the form of a tree, with highest level of abstraction in the knowledge representing the root nodes, and lower level of abstraction representing the branches from the root with the knowledge instances representing the leaves of the tree.

The concept of frames serves as a replacement for the means of representation of knowledge to that used in rule-based or fuzzy expert systems. But in order to enable efficient means of knowledge analysis, frame-based systems enable the incorporation of rules in the definition of frames. Thus, the inference mechanism in the frame-based expert systems utilizes a combination of both a rules and frames. The inferencing using

frames is achieved by a set of rules defined to evaluate the knowledge contained in the frames. The structure of these rules is identical to that defined in the rule-based or fuzzy expert system, however the processing of the rules relies on a pattern matching clause that enables the identification of the appropriate frame to operate on. In contrast to rule-based reasoning, goals in the frame-based reasoning can be both declared or evaluated through methods and demons. This provides the framework a dynamic outlook such that different inferences can be drawn based on varying states of the system.

In terms of the operational life cycle of the system, the basic steps remain identical to that of the rule-based expert system. The operation beings with the knowledge engineer defining the hierarchical structure of the knowledge and the creating the appropriate class-frames. The attributes for these classes are defined, and relationships are established between the various classes. Following this, instances are created in collaboration with expert engineers. The knowledge acquisition ends with the definition of the actions in terms of the methods and demons for the class attributes. At this point, the set of rules guiding the inference mechanism has to be defined. Following the acquisition phase, the system is evaluated and tested by expert engineers who tune, expand and revise the knowledge base so as to enable its usage by the target end-users.

The frame-based expert systems leverage the advantages of the frame data-structure of having an efficient and concise representation mechanism for knowledge. This representation scheme makes the evaluation of rules quite efficient in comparison to that of rule-based systems. But, on the other hand, the definition of the frame hierarchy requires considerable expertise from the knowledge engineer making the lead-time in the development of these expert systems higher. Owning to the nature of object-oriented

programming such an improper use of inheritances in such systems may lead to incoherent hierarchies.

2.3.4.4 Case-based Reasoning Expert Systems

The previously introduced classifications of expert systems were of the heuristic-based nature where the inference mechanism was driven by heuristic and domain facts explicitly defined in the knowledge base. An alternative to the explicit definition of heuristics can be achieved through the use of a data-driven inference mechanism. One example of such an expert system is the case-based reasoning expert system where the inference process in the system is driven by the identification of statistical patterns in observed data. In contrast to logical decision-making of rule-based systems, data-based approaches rely on statistical and probabilistic models enabling the generalization of the inference process across a variety of problem contexts. Systems that rely on case-based reasoning find application in a variety of fields ranging from entertainment and arts [104] such as music [105], computer gaming [106], to science and engineering [104] such as molecular biology [107], image classification and annotation [108].

The term case in case-based reasoning reflects experience. The experience in expert systems refers to the process of sequential decision-making undertaken by expert in the process of addressing a complex problem. The philosophy of case-based reasoning is the process of utilization of a collection of such experiences in the process of reasoning. Intuitively, these decisions made by such an approach are quite similar to the recognition-primed decisions where instinctive reflexes are trained and tuned by experience. Thus, decisions made by the such approaches are mathematically quite different from the logic-based reasoning as they do not stem from the flow of logic from true antecedent (or

assumption) to a true consequent (or conclusion), thus making them applicable in scenarios where antecedents and consequents do not hold a cause-effect relationship. In fact, it can be argued that the case-based reasoning framework emulates the instinctive nature of humans far better than the rule-based reasoning approach, as suggested by Minsky's Instinct Machine [25]. It will be argued that case-based reasoning provides a means for approximate reasoning where the similarity in the experiences is leveraged in the process of decision-making.

As indicated by its name, cases or experience form the fundamental entity in such systems. Cases form the underlying unit of knowledge stored in the knowledge base and all subsequent activities in the decision-making are driven by it. In general, each case represents a single instance of a recorded episode consisting of a sequence of decisions taken by a decision-maker. Cases are comprised of three components, a problem description, an associated solution and the outcome of the experience. The identification of the description of the problem refers to the generation of a series of attributes for the problem context, i.e., "the identification of those characteristic of the problem that are relevant and useful for solving the problem" [109]. Such a process, though, can be difficult for complex problem contexts. The solution in a case represents the decision (not the action) taken by the decision-maker in response to the problem. The underlying ideology of case-based reasoning is to repeat solutions that lead to a positive result while avoiding solutions whose outcomes are perceived to be negative. Thus, it is essential for each case to be associated with measure of the solution's merit, the outcome. Additional optional information can be associated with cases making it easier for the inference algorithm to identify relevant case to the problem at hand. This can be achieved by the

incorporation of meta-experiences such as rules to guide the use of a case, its frequency of usage, and so on.

Animals have a tremendous and unique capability to process natural language in the recognition of similarities between cases. In order to enable a computational system to reproduce such a capability, a level of formality is essential in the representation of a case. The purpose of the formality is to enable the estimation of a measure of similarity between the cases recorded in the knowledge base and the new one presented by the problem at hand. There have been several approaches established for the representation of cases depending on the data-structure of choice. The data-structures used for case representations include the feature vector or propositional representation where a vector of attributes describing cases that have no relation are stacked in a flat array, structured or hierarchical representation where cases are represented at different levels of abstraction and finally the unstructured representation for the use of representation of cases defined using text or images [110]. Depending upon the type of data-structure used different approaches to knowledge representation are chosen such as attribute-value [111] representation for flat array data structures, frame-based [112] and object-oriented representation [113] for hierarchical data-structures, and semi-structured and unstructured textual representation [114] for complex data-structures. Knowledge containers [115], attempt to merge the representation schemes with the similarity measurement, case bases and the adaptation rules to create a generic entity for the storage of knowledge in a case-based reasoning system.

The process of case-based reasoning is one that exploits the concept of similarities between cases in order to identify the stored case that is most similar to the case at hand.

The means of estimation of similarities between cases heavily depends on the representation chosen, but in most cases relies on the identification of the nearest neighbor to the new case. The process of design of the system relies on the identification of the appropriate representation scheme and the similarity measurement scheme would automatically follow. Schemes used for similarity measurement would yield an estimate of the distance between the two cases such that comparison between cases in the knowledge base can be perform for the identification of the appropriate number of neighbors.

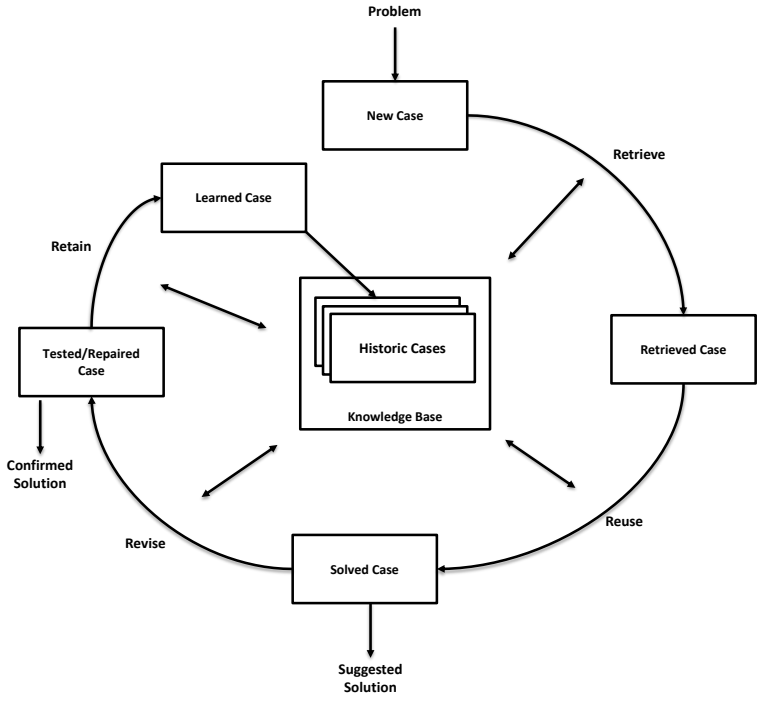


Figure 2.14: Classical operation cycle of a Case-based Reasoning System [109]

Figure 2.14 illustrates the classical model utilized by a case-based reasoning system in the process of problem solving where there are four primary steps identified in the process. The process begins with the system being exposed to a new problem. The system then utilizes the similarity measurement routine to estimate the similarity between the

new case and the ones in the database, so as to retrieve a set of the most similar cases. Having retrieved a selection of the most similar cases, the solutions associated with these are reused to propose a, perhaps new, solution to the problem at hand. This process may require adaptation of the existing solution in cases where no retrieved case is identical to the one at hand. The general concept of adaptation of a solution is illustrated in Figure 2.15. The proposed solutions are then evaluated in a revision step where the validity of the solution is evaluated which can be at the discretion of the decision-maker or it can be achieved computationally. This culminates with the implementation of a solution which results in the generation of a new case. Depending upon the performance of the new case, it may be incorporated into the knowledge base so as to guide future decisions represented by the retain step in the Figure 2.14.

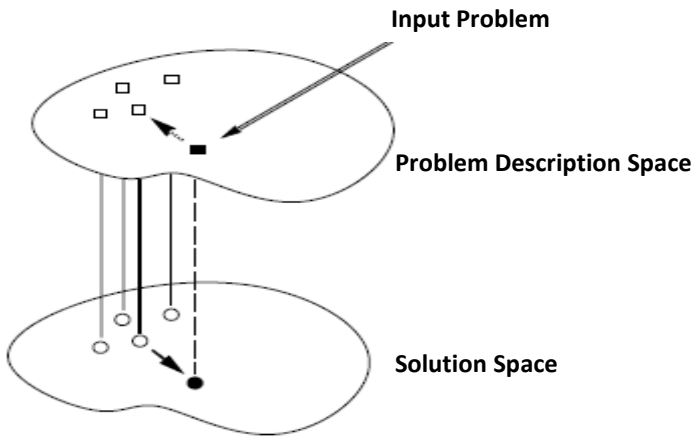


Figure 2.15: Generation of a solution in a case-based reasoning system [116]

One of the primary differences between rule-based methods and case-based methods is the ability of case-based methods to adapt to knowledge and learn from experience. The utilization of learning in reasoning forms a bridge between the branches of artificial intelligence where the expert systems are imbued with a capability for the

utilization of machine learning techniques in the process of decision-making. The learning process leverages trainable models and utilizes methods of supervised learning to adapt to the cases contained in the knowledge base thereby enabling generalization of cases across situations that hasn't been explicitly added to the knowledge base. The implementation of learning capabilities comes at the cost of computational expense. Training a model to perform satisfactorily in most cases requires a significant expenditure of computation resources and also expertise to achieve the correct configuration of the model. Further in order to train a satisfactory model, a significant amount of data may be required that prolongs the lead-time in the development of such systems.

2.3.4.5 Evaluation of the different types of Expert Systems

Section 2.3.2 identifies a set of desirable characteristics for expert systems. These characteristics are now used to evaluate the previously introduced classifications of expert systems in order to identify their suitability as a surrogate for expert engineers in a general sense. The results of this analysis are presented in Table 2.3 from which it is evident from the table that no single system performs satisfactorily in all categories, thus necessitating the development of a more generic framework capable of better representing expert reasoning.

Table 2.3: A comparative analysis of the existing alternative for Expert Systems

	Rule-based ES	ANFES	Evolutionary Fuzzy ES	CBR-ES
Adaptability	▼	▼	▼	▲
Inference Engine Development	–	▼	▼	▲
Uncertainty Tolerance	▼	–	–	▼
Learning Capability	▼	▼	▼	–
Explanation Capability	▲	▲	▲	▼
Maintainability	▼	▼	▼	–

2.4 Shortcomings of Expert Systems

In order to identify areas where the current research work is to contribute, an analysis of the shortcomings of traditional expert systems is carried out. The purpose of the analysis is to highlight portions of established development processes for expert systems that lack rigorous foundation or lack to adequately capture human reasoning. There are three aspects of expert systems considered in this analysis, the software development process utilized, the established process of knowledge acquisition and the inference mechanism utilized.

2.4.1 Development Process

The development process of an expert system is one that follows a software development process, an iterative and incremental development process of *evolutionary prototyping* [117] with both the waterfall and spiral development models being applied to the development of expert systems [118]. The development process of a knowledge-based expert system is highlighted in the Figure 2.16 where shows the iterative nature of

the development process where three different teams of engineers, the software engineers, knowledge engineers and domain experts, are involved. While there is no consensus on a single established development process for the lifecycle of an expert systems, most proposed processes [81], [119] include a common set of steps that represent the technical implementation of the expert system, the knowledge acquisition, the knowledge representation, the knowledge implementation and that of the verification and validation.

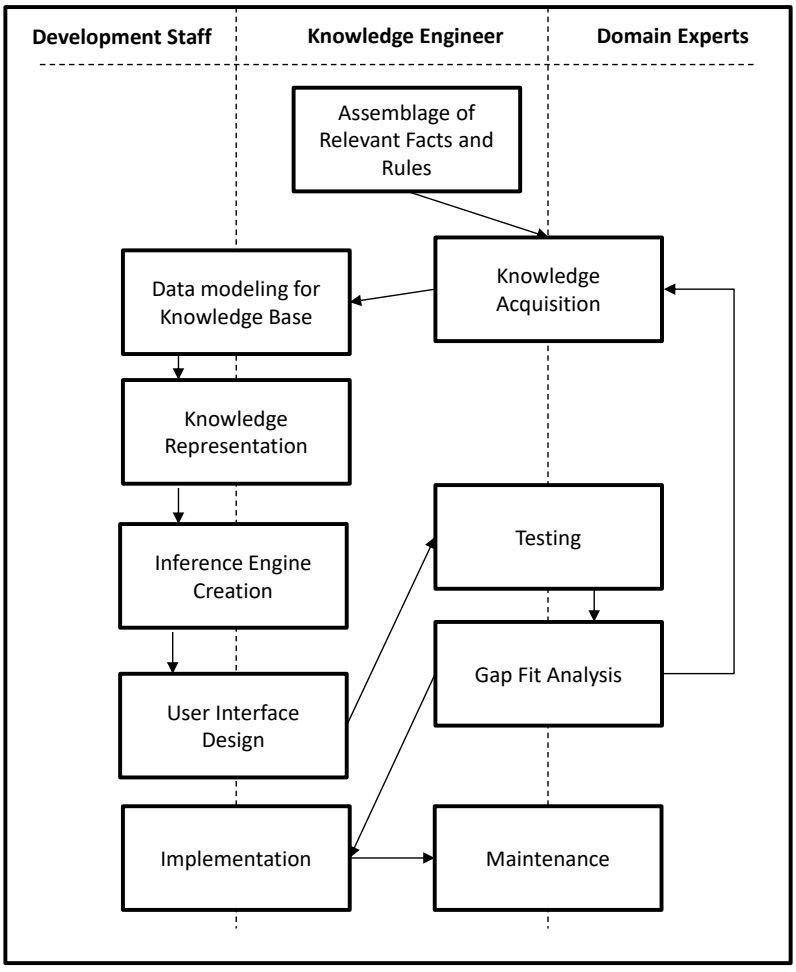


Figure 2.16: Lifecycle of a Knowledge-based Expert System [120]

The process of the development of an expert system begins with the identification of a problem context in which the system is required to operate. Expert systems are developed to operate within that particular scope to an extent that Waterman [81] defined expert systems as being applicable to narrow field in order to show reasoning capabilities similar to that of expert engineers. Having defined the problem and identified the necessary elements for the implementation of the expert system, an iterative process of configuring the knowledge base of the system is triggered. The process begins with the knowledge acquisition process where domain experts and knowledge engineers interact to identify elements of the domain knowledge that are relevant to the problem and is to be encoded in the system. Having identified a preliminary knowledge set, the process of data modelling and knowledge representation is triggered where the knowledge engineers collaborate with software developers to identify means of encoding the knowledge so as to be usable by the computational software. The step of knowledge representation is followed by one of implementation. Here implementation refers to the implementation of both the utilization of knowledge through the development of the inference system and the display means for the reasoning outcomes via the user interface. The development of the inference system is a process that is driven by the software engineer in collaboration with the knowledge engineer, while the development of the user interface is one that is driven by the software engineer in conjunction with the domain experts. The final step of the iterative process is the verification process in which the domain expert would test the completeness and validity of the knowledge base and also the inference mechanism. This steps involves the identification of shortcomings in the knowledge gathered so as to expand and revise the knowledge base and to tune and calibrate the inference system to account for the changes made.

Due to the iterative nature of the development process of an expert system, there is considerable lead-time in the system development due to the necessity for a collaborative environment involving different engineering teams, while having the entire system be applicable only to the identified problem context. This implies that in scenarios involving changes to design practices or design standards, the expert system would have to be redesigned which would involve repeating the entire iterative process. This leads to the first observation related to this iterative nature of the development of the expert system.

Observation 1
The expert system development life cycle follows process parallel to that of an iterative software development process involving several engineering teams to create a software product whose scope is limited to that of the identified problem.

2.4.2 Knowledge Acquisition

The purpose of knowledge acquisition is to convert the domain knowledge possessed by expert engineers to computational models that can be utilized in the expert system for the purpose of decision making. This process of knowledge acquisition and representation involves the knowledge engineers functioning as mediators between the domain experts and the software developers. The process of knowledge acquisition is typically implemented as an offline process in which the design engineers would interact with domain experts through a variety of means more often than not interactive and manual in nature. These include settings such as interviews, questionnaires, protocol analysis, interruption analysis or process replays [90], [121] each of which represent a manual interaction between the knowledge engineer and the domain expert.

Given that modern engineering problems are multi-disciplinary in nature, there is a considerable amount of domain knowledge that is required in order to create an expert system to represent the design process. But, as previously discussed, expertise is a rather rare commodity, even in large organizations, furthermore have access to expert engineers in each of the discipline constituting the design process would likely be even more difficult. Even with the assumption that there is sufficient access to domain expert at all times, and there is a sufficient number of domain expert available, the sheer volume of expertise that needs to be represented with the system to enable the usage for a complex design process would make the entire process time consuming. Finally, the process of knowledge encoding traditionally involves the hand crafting of features by knowledge engineers. If such a manual process has to be undertaken on large volumes of data that is needed to build an effective expert system, the resultant time in the development of the system would be quite large.

The above factors lead to what is known as the “knowledge acquisition bottleneck” [122]. This results in an extended development life-cycle for the expert system and has been observed as being the primary factor contributing to the lack of prolific use of expert systems in modern design environments.

This leads to the second observation, related to the process in which knowledge is extracted and encoded in the system. The offline knowledge extraction process could with the manual feature encoding results in the popularly known knowledge acquisition bottleneck that severely limits the application of expert systems to modern design environments.

Observation 2
Expert systems rely on the incorporation of knowledge in order to generate recommendations, but the process of incorporation of knowledge is the primary bottleneck in the development of such systems.

2.4.3 Inference Mechanism

The function of the inference system is to replicate human like decision-making so as to serve as a surrogate for expert engineers. Analysing the types of expert systems, the alternatives to the decision-making tactics are through the use of either decision rules or historical experience. The rule-based decision making, while not representative of the manner in which humans make decisions, was developed to serve as a replacement for the complex decision framework of the human mind. Rules represent the summarization of the guiding principles driving the decision-making process.

On the other hand, while experience-based decision making is representative of the manner in which humans undertake decisions, there is an inherent aspect of human decision making that is overlooked by the framework of case-based reasoning. This is the aspect of planning in decision making. Decisions made by humans, and animals, are a result of planning where long-term effects of decisions are considered. Each decision made is in response to a problem or a requirement and at every instant of time, a set of decision alternatives are evaluated and the one perceived to be the best is selected.

An inherent flaw in both approaches is their inability to plan. The established frameworks rely on the instantaneous knowledge and problem context at hand in order to make a decision that is considered optimum for that state without any consideration of

future impacts of such a decision. This is due to the lack of a sense of a long-term value of the decision. The long-term value associated with decisions have been seen as the primary drivers in the decision-making in animals. In fact, it has been argued that the primary function of the prefrontal cortex in humans is to account for planning and long-term decision making [123], [124]. It has to be noted that the process of planning involves the estimation of values of the decision alternative, and these values are often uncertain in nature. This could be a result of the nature in which the decisions manifest into actions, wherein actions can result in an uncertain effect on the system.

This leads to the third and final observation about the shortcomings of traditional expert systems. The inability to plan results in decisions made based on immediate returns which is not a true representation of the nature in which humans make decisions. This inability to plan results in situations where long-term plans have to be accounted for in the representation of the knowledge and associated inference mechanisms utilized, an additional consideration that is to be made by the knowledge engineer in collaboration with the domain experts.

Observation 3
Expert systems are only capable of making decisions that have immediate returns but are incapable of replicating animal behavior of long-term planning, a scenario in which impacts are often uncertain.

CHAPTER 3. RESEARCH METHODOLOGY

The review of the structure of and the types of expert systems in CHAPTER 2 concluded with the identification of a set of issues, that prevents the acceptance and usage of expert systems in most modern design environments, resulting in a set of observations regarding the state-of-the-art for expert system development. Guided by these observations, the current chapter aims to establish a research methodology so as to address the shortcomings identified in the observations. In the way of developing a methodology, first, the observations are formalized to generate a directed motivation for the research from which the research goal is established. The research goal is then decomposed to identify the objectives of the research work which in turn lead to the hypothesis and the questions guiding the research work. A plan for the validation of the research work is introduced in terms of engineering applications and finally, an analysis plan is introduced to evaluate the capabilities of the output of the methodology.

3.1 Research Motivation, Goal and Objectives

3.1.1 Research Motivation

The problem of engineering design has been identified as being one of sequential decision-making [59]. This is evident in all phases of design, be it the conceptual design, where decisions are less expensive but made affect an abstract representation of the product, or the detailed design, where the process involved in the decision making are costly but the return on the decisions made are more significant. Further given that the engineering design process is one that is often standardized and follows a well-established set of protocols, there is a significant room for automation in such practices,

its key hurdle being the necessity for engineering judgements and domain expertise. As a result of the inaccuracy in physics modeling, insufficient computational capabilities or just an imperfect understanding of the governing physics, expert knowledge and judgement is an irreplaceable commodity in modern design processes and environments.

As indicated in the introduction to expert systems, the purpose of expert systems is the utilization of human knowledge in the replication of the decision-making resultant from it. But, through its evaluation, there have been several shortcomings identified in the adopted development and implementation process. These shortcomings make such systems inefficient in or incapable of replicating human-like decision making and, thus, their application to real-world engineering problems infrequent. The field of engineering relies on the availability of mathematical guarantees before the adoption of certain practice or process. While expert systems, traditionally, utilize the logical reasoning paradigm, that has its foundations in sound mathematical principles, it is almost impossible to represent the vast variety of engineering cases in such a format. This often results in expert systems being designed for applications that are but a semblance of the true engineering problem representing a narrow sliver of the original problem.

The ever-improving computational capabilities have resulted in a change in the paradigm of design engineering where there is now,

- the utilization of advanced design processes such as numerical simulations and statistical methods
- reliance on third party engineering application providers in lieu of in-house custom designed applications for engineering

- the incorporation of multi-disciplinary analysis techniques and the development of integrated design and analysis environments

As a result of this change in the design environment and design paradigm, a traditional outlook to the development of expert systems would not suffice. With such a perspective the current research work draws motivation to address some of the identified shortcomings of expert systems so as to make them reasonably applicable to the modern engineering design paradigm.

Research Motivation
The current research work is aimed at the development of a framework for knowledge-based expert systems that enables its application to modern design environments by, • Improving the process of knowledge acquisition with the removal of the offline and manual knowledge acquisition process. • Improving the process of autonomous decision-making by enabling capabilities such as learning and planning. • Developing a generalized framework for the resultant system such that it is applicable to a wide-variety of problems and processes.

3.1.2 Research Goal and Objectives

The formalization of the research motivation is carried out with the development of a research goal. The research goal highlights not only the expected output of the

research process, but also the expected benefits from the research work. The research goal for the dissertation is stated as,

Research Goal
The research work aims to develop a framework that enables the extraction of knowledge from a design system such that an intelligent agent can learn to mimic the extracted behavior with an ability to self-learn in order to assist design engineers during the usage of the design system so as to develop products with shorter time to market, improved process efficiency and improved product flexibility and quality.

The goal for the research work is rooted in the development of a comprehensive framework to enable knowledge-based decision making in engineering design. While the goal does not yet address any of the shortcomings of expert systems, it does note that the developed framework has to enable knowledge-based decision making. Thus, such a framework would serve to replace traditional knowledge-based expert systems but is primarily targeted at the engineering design problems. In order to achieve the above stated goal, a set of three objectives aligned with the "gaps" observed are identified. This is illustrated in Figure 3.1 where a set of research objectives are developed from the previously observed gaps, thereby leading to the identification of an area of research corresponding to each objective from which methods and capabilities are borrowed upon for the realization of the objective.

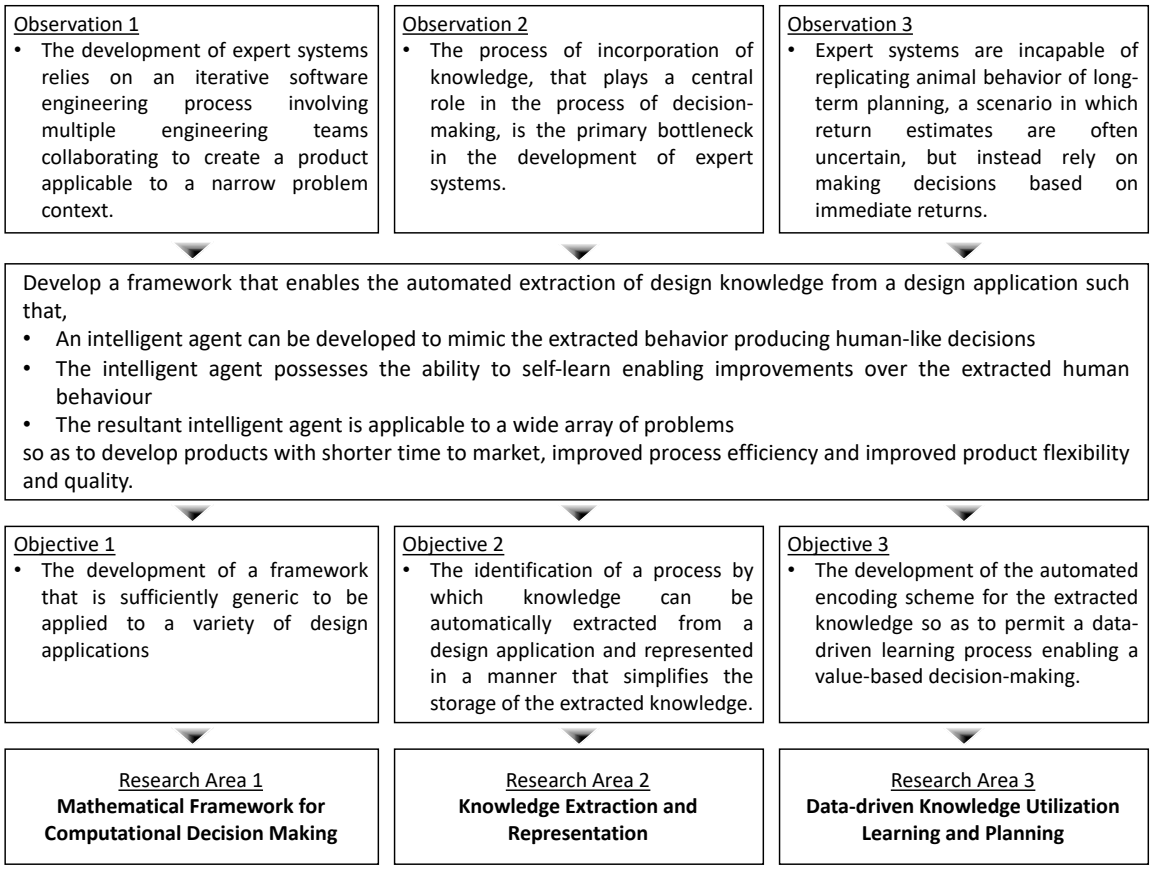


Figure 3.1: Mapping of observed shortcomings to the research goals, research objectives and the areas of research

1. First, the framework has to be capable of being applicable to a wide variety of problems and should be capable of handling dynamic nature of the engineering design process. Thus, there is a necessity for the development of a generic framework capable of being designed on one but applied to other problems as well. Due to the nature of engineering, such a framework should rely on sound mathematical principles in the process of knowledge extraction, representation and decision-making. It is claimed that if the developed framework decouples the process of knowledge extraction, representation and that of the decision-making from the actual knowledge being extracted, represented and utilized, then the

resultant framework would be applicable to any problem, regardless of its nature or complexity.

2. The second observed gap in the expert system corresponds to the manner in which knowledge is extracted and represented even in the state-of-the-art systems. When a designer (expert engineer or otherwise) exercises a design process, all the necessary knowledge about the process is exposed to the design system in one way or another. In such a scenario, it stands to reason that an offline process of knowledge extraction would be unnecessary provided that a means of identification of the necessary knowledge entities can be established and an associated means of representation of the identified knowledge can be developed. This leads to the second research objective in which the target is to identify a means to extract and represent engineering knowledge directly from the design process in an automated fashion without the reliance on an offline knowledge acquisition process.

3. The final observation related to the manner in which the inference mechanisms operate in traditional expert systems. In particular, the lack of the ability to plan in established approaches to inference. The objective stems from the assumption that an imitation of human-like decision making in a computational framework results in replication of human-level results. Thus, the third and final objective of the research work involve the identification of means to replicate human-like decision making. In other words, enabling the capability for planning based on experienced behavior of the system.

3.1.3 Generalized methodology guiding the development of the framework

In contrast to the existing approach for development of expert systems, the current research work attempts to develop a framework and not a software application. The purpose of the framework is to enable the utilization of any existing engineering design application and imbue them with the capability for knowledge-based decision-making so as to either train, assist or replace novice users of the design application in problems related to the knowledge extracted. Although, the methodology utilizes a well-established process in the expert system development life cycle, an attempt to generalize it across all problem contexts is made. Figure 3.2 illustrates the generalized process for the framework development and is comprised of the "three K's", the knowledge extraction, knowledge representation and the knowledge utilization, driving the development cycle of the framework. At all times during the development of the framework, the target of enabling the utilization of the generated knowledge from a design system or process is considered and hence the steps are inherently designed to generate the knowledge in a fashion optimized for the purpose of utilization for decision-making. It is argued that the implementation of the "three K's" would yield a framework that can enable knowledge-based decision-making for any design application.

Figure 3.2: The "three K's" in the development of the knowledge-based decision-making framework

3.1.3.1 Knowledge extraction

The process of knowledge extraction refers to the automated extraction of pertinent data and decisions related to the design application. The primary necessity for the generalization of the knowledge extraction is that there should be no human input in the knowledge extraction process thereby making the entire process generic and automated. This places rather strict requirements on the design application and design processes that can be considered as being suitable applications for the expert system, addressed in 3.3.

The assumption of the application of engineering design as the target application directs the type of knowledge that is to be extracted. Each engineering design process is driven by a set of requirements and the design process is carried out in a software application whose primary function is the representation of the design and its performance. Thus, the necessary information to be extracted from the system/process at any instant of time would be the requirements driving the design process and the visible representation for the design.

3.1.3.2 Knowledge representation

The task of knowledge representation refers to the identification of appropriate means of storing of the extracted knowledge for the downstream decision-making process. As the extracted knowledge would consist of requirements, design representation and design performances the representation scheme chosen would likely have to account for differences in the type of extracted knowledge. For example, requirements are typically specified in natural language, while the design representation would typically be parameterized in terms of a set of key design features.

The constraint placed on the knowledge representation is that the generated representation scheme is required to dynamically adapt to a growing knowledge database. Owing to the dynamic nature of the knowledge base it is certainly a possibility that the representation data are subject to change. Thus, any preconfigured representation scheme would not be suitable as it would be unable to adapt to the dynamic nature of the design environment, which in turn makes the process of knowledge representation generic.

3.1.3.3 Knowledge utilization

The manner in which humans utilize knowledge is two-fold, first in the process of decision-making where knowledge is utilized to make inferences, drawing conclusions and reasoning about a certain problem and secondly in the process of learning, where knowledge about a problem is utilized to derive strategies, decision rules and often aiming at the optimization of the result of the decisions emerging from the knowledge gathered, i.e., identifying how and when to best use the knowledge gathered.

In a computational sense, the process of knowledge utilization refers to the identification and exploitation of patterns in the extracted knowledge thereby guiding the decision-making process. While there are several established means of utilizing extracted knowledge, most rely on approaches that are not replicative of human-like decision making. In order to reproduce such a capability, the utilization scheme chosen should be capable of not only learning from the demonstrated knowledge, but also acquiring additional knowledge to further influence the learning process. Another aspect of the utilization of knowledge is the conversion of the extracted and represented knowledge into a format that enables the application of the inference mechanism for decision-

making. This refers to the problem of knowledge encoding and is dependent on the utilization method chosen.

In order to generalize the knowledge utilization scheme, the constraint placed on the selection of a scheme is that it should first and foremost be capable of imitating the demonstrated human behaviour, i.e., reproduce human-level decision-making in an automated fashion. Further, the utilization scheme should be capable of automatically adapting to changes in the knowledge extracted so as to refine decisions as and when new information is available. Finally, the entire process of decision-making has to be founded on mathematical principles, i.e., data-driven, that can guarantee the choice of appropriate decisions based on the knowledge available. These requirements placed on the knowledge utilization scheme ensure that the selected process would be generic enough to be applicable to any application, as it would be data-driven with the data being generated by the knowledge extraction and representation processes.

3.2 Research Questions and Hypothesis

In order to guide the research work carried out, a set of three research questions are formulated, each associated with an objective and the corresponding area of research. The research questions, while being abstract, provide insight into the capabilities that have to be used to realize the goal of knowledge-based decision-making for engineering design. Thus, they help identify the key enabling methods that a researcher would require to enable the implementation of a generic framework capable of incorporating knowledge in the process of artificial decision-making.

3.2.1 First Research Question

The first research question deals with the identification and selection of a mathematical framework that is suitable to address the problem of decision making in engineering. If one such framework can be identified, its computational equivalent is required to be developed. With the implementation of the computational approach for solving the decision-making in engineering domain, it is required to evaluate if the capability can be applied an entire engineering design process. The application of the problem to engineering design is a highlight of this research question as it scopes the focus of the research work purely to workflows, processes and applications used in the field. These issues are formalized in the form of the first research question, given below.

Research Question 1
What are the desired characteristics of the framework that enables the utilization of knowledge in the process of addressing an engineering design problem? • What are the available mathematical tools that reflect these characteristics? • What is the computational representation of such a mathematical tool? • Can a computational implementation of such a framework be applied for general purpose engineering design?

These set of questions address the first objective, i.e., the creation of a generic framework that addresses the abstract problem of decision-making in engineering design without any constraints on the engineering application. The computational representation of the framework is prioritized as it is imperative that artificial agents are created so that

design processes can be improved upon by leveraging the sheer power of computing. In addition to the necessity for a computational implementation, its mathematical background stresses the importance of theoretical backing for the process of decision-making, a criterion that is a necessity in the field of artificial decision-making. Given that artificial agents are designed to recommend to and take the place of design engineers, the presence of a sound mathematical background ensures adequate justification for the decisions made by an agent.

An answer to the first research question can be established through first, an identification of the key characteristics of engineering design. These characteristics can establish the desired capabilities that have to be provided by the mathematical and computational frameworks. Upon identification of these capabilities, a review of the established mathematical methods should identify one or more suitable candidates that also provide sufficient mathematical justification. Following an evaluation of the identified candidates and the selection of one of them, a computational representation for the mathematics has to be identified and implemented. The identification of the means of a computational implementation can be accomplished by a review of the literature, while its implementation would require the development of the methods and algorithms identified. A proof-of-concept can be utilized for the validation of the computational implementation and to also evaluate its applicability to the field of engineering design. The description of the proof-of-concept is addressed in 3.3.

3.2.2 Second Research Question

The second research question addresses the area of knowledge extraction and representation. In particular, it deals with the identification of the knowledge that is to be

extracted from the design application, the means for doing so and also the identification and implementation of the means for representation and storage of the extracted knowledge. The research question builds on the findings of the first research question. In fact, it is posed in an abstract sense so as to help identify the necessary knowledge to uniquely represent the design application and/or process. The research question also deals with the issue of knowledge representation where a means for the storage of the knowledge is required to be assessed. In order to ensure the generalized nature of the process, no assumptions are permitted on the nature of the knowledge that can be extracted. These issues are, again, formalized in the form of the second research question, given below.

Research Question 2
In the presence of a design environment, how can extract knowledge in an automated fashion so that an adequate representation of the design application can be generated? • What does it mean to adequately represent the design application, i.e., what is the knowledge that is to be extracted from a design application to completely and uniquely represent it? • In a dynamic setting where knowledge is extracted continuously resulting in changes to the knowledge database, how can extracted knowledge be stored?

The above question addresses the second objective, i.e., the development of an automated online approach to the problem of knowledge acquisition. The research question prioritizes the presence of a dynamic knowledge base in order to prevent the

hand-coding of the representation there by enabling the automation of the knowledge extraction process. Further, to ensure that any representation chosen is “discovered” by an algorithm a dynamic nature of the knowledge database is assumed. While the premise of the research question, the automated online knowledge extraction, guarantees a dynamic knowledge database it is still stated as an assumption as a solution to the research question is still pending.

The answer to the research question can be realized through a set of two experiments. There is, in fact, a necessity for two experiments in order to validate the generalization capability of any developed process. While the answer to the sub-questions, i.e., adequate representation of the design application and dynamic adaptation to the stored knowledge, are dependent on the solution to the first research question, general guidelines can be established for these. First, in order to generate an adequate representation of the system, every generated representation has to uniquely describe the system at that instant of time. This can range from a vectorized representation of the product being design to an image representation of the active design application screen. As these are application dependent, the issue is addressed further in Section 3.3 when the problem scope is introduced. Second, in order to adapt to the dynamic nature of the knowledge, the representation scheme should either represent the entire state of the application at every instant of time or dynamically adapt the representation to correspond to changes in the state representation. Since a restriction is placed on the involvement of humans, it is not possible to specify the necessary representation a priori. Thus, the generation of a complete representation of the system at the start of the design process would be impossible, which leave the option of a dynamic adaptation of the representation scheme.

3.2.3 Third Research Question

The final research question deals with the implementation of a mechanism for the utilization of the extracted knowledge to enable the process of decision-making in an artificial agent. The research question is framed so as to identify an appropriate algorithm that enables an agent to not only learn from expert behaviour but also, self-learn through exploration. In particular, it deals with address the question of learning from a combination of both expert demonstrations and acquired knowledge. A consideration inherent in this is the encoding scheme utilized for the expert demonstrations that is suitable for the application of the learning mechanism. These concepts are formalized in the form of the third research question, given below.

Research Question 3
How can the combination of demonstrated data and experience data be used to train an agent to make effective decisions? • How can the extracted knowledge be encoded so that learning techniques can be applied? • What is the process that enables the encoding of different sources of knowledge? • In the presence of a dynamic knowledge database, how can an agent adapt to changes in the knowledge database? • In the presence of both demonstrated and experienced data, what are the necessary modifications to the framework so that a hybrid learning strategy can

be utilized?

The research question posed above addresses the third objective, i.e., imbuing a learning capability to the inference mechanism. Prior to enabling learning in an artificial agent, any knowledge acquired has to be encoded in a means that is suitable for the learning algorithm. This first requires the identification of a learning algorithm which would establish the encoding format necessary. A process for the encoding would then follow, with the consideration that knowledge can be in different formats and can be gathered from multiple sources. Following this, the basic necessity of being able to learn from expert demonstrations is framed with the goal of replicating human-level performance so that the artificial agent results in performance identical to that of the demonstrations. This is then expanded so that a capability for the improvement over demonstrated performance is evaluated. The purpose of this formulation is to capture scenarios where expert demonstration as insufficient to find the optimum sequence of decision in the design problem. In such cases, the artificial agent would need to outperform the human equivalent through the utilization of an exploration policy where alternative decision-making policies are explored. Finally, due the presence of both expert demonstrations and explored knowledge, the presence of a suitable means of mixing these different sources of knowledge is evaluated.

The research question can be addressed in two sections, the first dealing with the identification of the learning strategy and the second its implementation and evaluation. A survey of the literature related to the identified mathematical and computational framework should reveal suitable alternatives of algorithms for the implementation of the learning capability. Since the second part of the research question primarily deals with

the evaluation of the performance of the learning capability, a series of analysis experiments can be devised to perform such an evaluation. The details of the experiment are addressed in 3.4. A prerequisite for these experiments, though, is the presence of a framework capable of knowledge acquisition both from expert engineers and artificial agents. Thus, any experiment devised would build on the solutions to the previous two research questions.

3.2.4 Overarching Research Hypothesis

Having established the questions to guide the research work, a formal hypothesis establishing the central theme for the dissertation is presented. It is hypothesized that there exists an approach that can be leverage so that knowledge can be extracted and utilized from generic design applications such that artificial agents can be utilized to reproduce human-like decision making based on the knowledge acquired. This is stated formally in below.

Hypothesis
• It is possible to develop a process that enables the extraction of expert knowledge from most design applications. • Having extracted the necessary knowledge sound mathematical principles can be exploited for the reproduction of expert-like decision-making in the very same environment, through the use of an artificial agent. • The creation of an artificial agent would in turn enable the automation of decision-making enabling a capability of self-learning, i.e., learning through

exploration.

- In scenarios where expert demonstrations are suboptimal, a self-learning agent would be able to improve upon the demonstrated behaviour.

3.3 Research Scope

This section scopes the research work by identifying a set of applications and experiments which help in answering some of the research questions previously introduced. In particular, the section identifies a set of desirable characteristics from the design applications to which the process of automated knowledge extraction and utilization is to be applied.

In identification of appropriate characteristics of design application, first an analysis of the types of design applications is to be performed. In terms of the visibility of the design application, each application can be categorized as falling somewhere along a spectrum of applications, with the spectrum denoting the visibility of the system. The two ends of the spectrum are denoted by black-box systems, where there is no visibility into the operations of the system and all interactions by an engineer are restricted to the presented interface, and white-box system, where there is complete visibility in the operations of the application and any information contained in the application can be queried at any instant of time. This categorization of design application is based on the manner in which interactions are undertaken with a design application, illustrated in Figure 3.3.

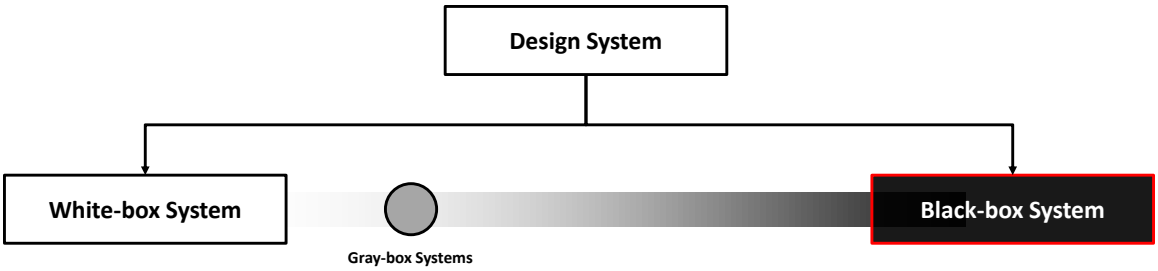


Figure 3.3: Categorization of design applications based on available interaction mechanism

As illustrated in Figure 3.3, white box systems offer complete control to the engineer where all the necessary information regarding the product being designed or the design process undertaken can be queried in order to populate a knowledge base. In such a system, there is a clear visibility of each action undertaken by a designer indicating a decision made, and the effect of the action can also be ascertained. Such applications include open-source frameworks and in-house codes that permit an automated callback for the extraction of knowledge from the design application. Moving along the spectrum towards the other end, one would encounter grey-box systems that are closer in nature to the white-box systems. In such a system, while a designer may not have complete freedom to query any knowledge imaginable, the system does provide a means to query the entire state of the design process or the product. Further, such system does indicate human actions so that an analysis can be performed to identify the type of decision made. An example of such an application would be an MBSE framework that supports event notifiers. Moving further along the spectrum close to the black-box applications, one would encounter applications which support a capability for automation but fail to reveal any information about the manner in which the design process or product is represented. Such applications are quite similar to the black-box systems where user interactions are primarily carried out through an interface, but there is the possibility of a programmatic

extraction of information enabling the automation of the system. Examples of such system include commercial and free software that support automation through an API such as Siemens NX, ANSYS, etc. The final variety of system are the complete black-box systems whose only means of interactions are through the established user interfaces and the supplied user entry points. The dissertation work does not address the problem of a generic black-box system and the extension of the framework presented in the current work is addressed in the future work section in CHAPTER 8.

Thus, based on the above-mentioned categorization of the design applications, a set of two use-cases are developed to evaluate and demonstrate the applicability of the framework to a range of design applications. The developed processes for knowledge extraction, representation and utilization are to be maintained the same across each use-case. This in turn establishes the desired characteristics for the design applications that can be considered for the application of the framework, shown below.

- It has to be possible for an external process to connect to the design application to extract knowledge from the system.
- It has to be possible for an external process to connect to the application in order to automate its behaviour.
- The application has to provide some means of indication that a user-interaction has occurred.

3.3.1 Use-case 2: A Design Application implemented using principles of Model-based Systems Engineering

The second use-case deals with the application of the automated decision-making framework to a grey-box system. The use-case is applied to the architectural and conceptual design of an unmanned aerial vehicle for a undergoing a fixed design process. In order to enable the use of such an application, a model-based systems engineering application is developed such that the process of conceptual design from the definition to the verification requirements can be carried out within a single integrated environment. The nature of model-based systems engineering is such that the state of the active design is represented in terms of a model which is an object-oriented representation of the design. In addition, these framework enables the definition of requirements in natural language that serve as indicators for the performance of the design. Finally, a signal-slot paradigm for event notification is adopted such that decisions made by the user can be traced to identify and evaluate them. Thus, while the entire application is developed in-house, from the perspective of the framework, it is treated as a grey-box system. The details of the application and the associated use-case are discussed in CHAPTER 6.

3.3.2 Use-case 3: A black-box System with API access

The final use-case deals with the application of the framework for the purpose of automation of a generic commercial engineering design application. For the purpose of the dissertation the investigation is limited to Siemens NX. In contrast to other case considered, Siemens NX there are multiple sources from which knowledge can be extracted from the Siemens NX application. Thus, the framework would have to be configured to be able to handle the incorporation of multiple sources of knowledge.

Further, the representation of the design product within the application is not visible to the user. Instead a select few attributes and properties are exposed to the user via the application's API. Thus, the type of knowledge that can be extracted from the application is restricted by the information that can be gathered. In the application of the framework to the final application, there is not one particular design product that is considered, but instead it is desired to capture patterns in the operation of the application that result from the specified set of requirements. In contrast to the other applications, the goal of the framework when applied to Siemens NX is scoped to the imitation of human-like behaviour and not improvement of it. The details of the configuration of the framework in order to enables its application to the large-scale problem of Siemens NX is discussed in CHAPTER 7.

3.3.3 Summary

Thus, owning to the complexity in the development of a framework that is capable of automated decision-making and is generic enough to be applicable to a variety of engineering design applications two use-cases established. Each of these use-case serve to answer a portion of the second and third research questions, mainly the process in which knowledge is extracted, represented and utilized and the framework for doing so. While these use-cases serve to demonstrate the ability for the automated decision-making they do not evaluate the extent of these capabilities, i.e., they do not serve as a stress test evaluating the extent to which the capability can be exploited. These are discussed next where a set of analysis experiments are devised to assess the computational performance of the framework.

3.4 Analysis of the decision-making capability of artificial agents

While the use-cases previously established define the problems on which the framework is tested, the analysis experiments establish the suitability of the framework as a replacement for established approaches to knowledge-based decision-making. In order to draw such a conclusion, the artificial agent enabling the knowledge-based decision-making has to be able to first and foremost, imitate human-like decision-making and then outperform it in cases where human-like decision-making is suboptimal. Finally, the capability of the framework to adapt to changes in the problem context needs to be evaluated to estimate its robustness to changes in the learning conditions. These settings are illustrated in Table 3.1 and these experiments are carried out on the second use-case of UAV design. As depicted in Table 3.1, the experiments of imitation and improvement can be viewed as satisfying the necessary and sufficient conditions for a large-scale development of the framework while the adaptability experiment can serve to provide insight into the framework and processes' robustness. The details of the experiments and the setup of the analysis are discussed in CHAPTER 6.

Table 3.1: Experiments planned to evaluate the performance of the developed framework

Experiment	Condition	Criterion	Evaluation Metric
Replication	Necessary Condition	Can an artificial agent learn to perform as well as a human-operator in the	• Output performance of the design or designs selected. • Amount of learning steps required.

		presence of human demonstrations?	• Amount of demonstration data required.
Improvement	Sufficient Condition	Can the artificial agent learn to perform better than the human demonstrations through explorative schemes?	• Output performance of the design or designs selected. • Amount of exploratory data required. • Amount of training steps required.
Adaptability	Robustness Condition	Can the artificial agent apply learnt knowledge to different problems, without additional training?	• Output performance of the design or designs selected. • Estimate of similarity between the problems.

Table 3.1 (Continued)

CHAPTER 4. RESEARCH BACKGROUND

The previous chapter established a methodology for the research work by identifying a set of guiding research questions. The current chapter addresses these questions by establishing a hypothesis for each of them. The hypothesis in the current chapter is driven by a survey of the available literature and concludes with the identification of a concrete plan for the implementation of the knowledge-based decision-making framework. The chapter is structured such that each research question is addressed in sequence with the investigations into the later questions building on the hypothesis developed in former ones.

The first research question investigates the feasibility of utilizing existing mathematical tools for the automation of an engineering design process and then looks into the computational representation of one such mathematical framework. The computational representation dictates a series of necessities in terms of the knowledge extraction, in particular, it answers the question “What is the knowledge that is to be extracted?”, but there is freedom for the means in which the knowledge is extracted and, also, represented. Finally, the computational framework, also, establishes a set of guidelines for the manner in which the knowledge is to be encoded so as to enable data-driven learning and the possible learning schemes are investigated in the final third of the chapter.

4.1 Research Area 1: Mathematical framework for Computational Decision Making

As previously indicated, the purpose of the first research question is to ensure that the adopted process of decision-making relies on sound mathematical principles. Prior to the identification of an appropriate framework that enables the process of autonomous decision-making, it falls upon the author to justify the capability of representation of an engineering design process as a mathematical problem. This is addressed by analyzing a handful of characteristics of the engineering design process with the hopes of relating them to that of autonomous decision-making.

4.1.1 Engineering Design

A comprehensive review into the concept of a design is provided by Ralph and Wand [125]. In their review, several different definitions of the concept of design are analysed and a proposal for a formal definition is made. The definition, given below, establishes a formal set of interacting components, illustrated in Figure 1, that help identify some of the basic elements leading to the creation of a design and thus establishing the context for engineering design and the engineering design process.

Design

A design is the specification of an object, manifested by some agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to some constraints.

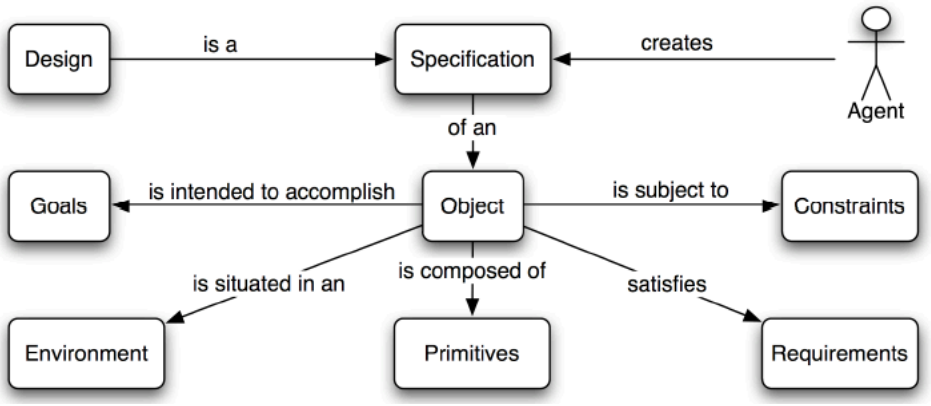


Figure 4.1: Conceptual model of a design [125]

Based on the generic definition given in Figure 4.1, with the application of an engineering context, one could arrive at the definition of engineering design by adding on the requirements for the use of scientific, technical and/or statistical knowledge in the production of the design [126]. Several models have been proposed to outline to process in which engineering design is carried out. For example, an abstract view of the process of engineering design is provided by the iterative interactions between the problem identification, information gathering, alternative generation, alternative evaluation and the testing and implementation steps, illustrated in Figure 4.2, while the details of each of the individual step is omitted in favour of abstraction.

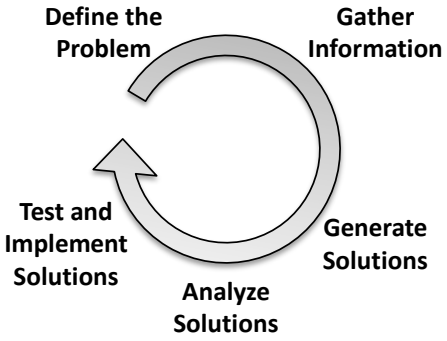


Figure 4.2: Abstraction of the iterative engineering design process [127], [128]

A more detailed representation of the steps involved in the design process is provided by Dieter [129], illustrated in Figure 4.3, provides a comprehensive overview of the entire process. The process is divided into three phases, the conceptual design, embodiment (or preliminary) design and that of the detailed design. The conceptual design revolves around the generation of a set of feasible and viable design concepts that are capable of addressing the customer requirements. The set of designs generated in such a setting remain abstract in nature and their analyses are, typically, performed at a lower fidelity. Having identified a set of feasible designs, the phase of embodiment design involves the generation of a physical layout for the conceptual design. Along the process, component models are utilized to ensure each subsystem or component of the system performs to the specified tolerances thereby ensuring the given customer requirements are satisfied in the presence of increased analysis fidelity. The final phase of detailed design phase involves the generation of part specifications for the manufacturability of the constituent components. This involves the generation of 3D models or detailed drawings while accounting for both an improved fidelity in any physics-based analysis and constraints introduced by the available manufacturing processes.

An important aspect of the above detailed description of the engineering design process is necessity for decision making at every step of the process. It has been argued [59] that the entire design process can be represented as a sequence of decisions. Thus, in order to replicate the decisions made during design, it is necessary to incorporate the field of decision analysis in any considerations made.

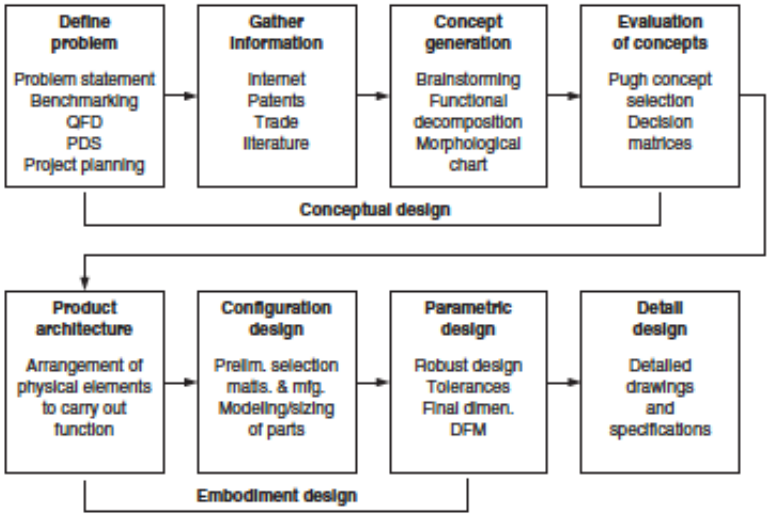


Figure 4.3: The three phases of the engineering design process [129]

4.1.2 Sequential Decision Making

The problem of sequential decision making [130] is one that has been studied in great detail with well-established approaches being readily available to address this task. Typical approaches rely on the framework of Markov decision processes [131] that necessitates the presence of a model of the decision making environment. The Markov decision processes are an extension of the Markov chains and the Markov process which build on the concept of the Markov property. The Markov property can be stated as follows:

The state of the system is only dependent on its most recent state, and not its history of states.

In addition to the Markov property, the Markov chains assume that there is a time independence in the event being modelled. This implies that transitions between states are independent of the instantaneous time of the system. With these fundamental

assumptions, a Markov Process [132] can be defined as a mathematical model of a system in which the system dynamics is represented by a series of "states" and "transitions" where the variables of the state provide a mathematical description of the system and the transition accounts for the changes in these variables.

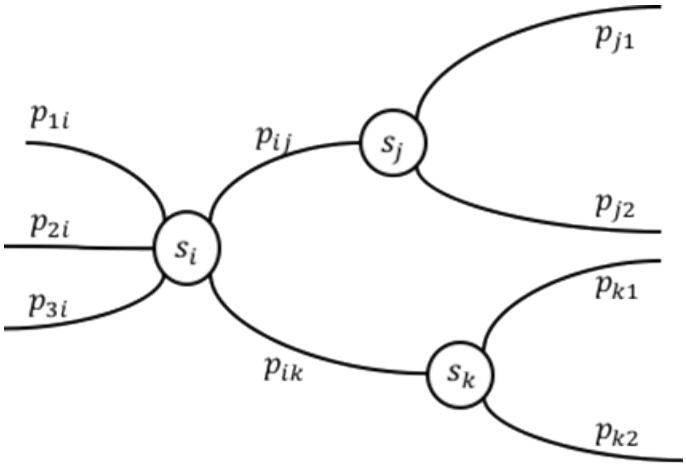


Figure 4.4: An illustrative example of a Markov Process

The Markov Process is probabilistic in nature in the sense that the transitions from one state to another is dictated by the system model and is mathematically represented by the transition probabilities. If each transition is assumed to occur at a discrete time point, say t_i, then the transition from state i to j is purely a function of the probability at state i and not any previous state. Figure 4.4 provides an illustration of a Markov Process in which the states are represented within the ellipses and defined by the set of state variables **s**, the transitions are indicated by the arrows connecting the states and the transition probabilities by p_{ij} such that the probabilities satisfy the conditions,

$$\Sigma p_{ij} = 1 \text{ and } 0 \leq p_{ij} \leq 1$$

Another component of Markov Processes is the concept of "rewards". A reward is indicative of the returns observed when the system transitions from one state to another. These rewards are, thus, random variables whose distribution is governed by the system dynamics, i.e., the transition probabilities. The reward parameters play an important role in the field of computational decision making as the goal of such a framework would be to identify the sequence of states that maximizes the expected cumulative reward observable to the system. The established approach for the computation of decision is through the use of dynamic programming and Bellman equation [133].

The mathematical representation of a system whose states are affected by an external operator, i.e., an agent is modified from the standard setting of the Markov Process to incorporate the concept of "actions". An action represents the interactions an agent can have with the system in order to affect, or alter, the system's state. The mathematical representation of the system[3] is termed as the Markov Decision Process [131], illustrated in Figure 4.5, and is the accepted standard for solving sequential decision making problems [134], [135].

Markov Decision Process

A Markov Decision Process is represented by the tuple $< S, A, T, R >$ *where* S *is the finite set of states,* A *the finite set of actions,* $T: p(\bullet | s, a)$ *represents the state transition probability and* $R: r(s|a)$ *is a reward function.*

[3] In some future occurrences of the term, the terms environment and system are used interchangeably.

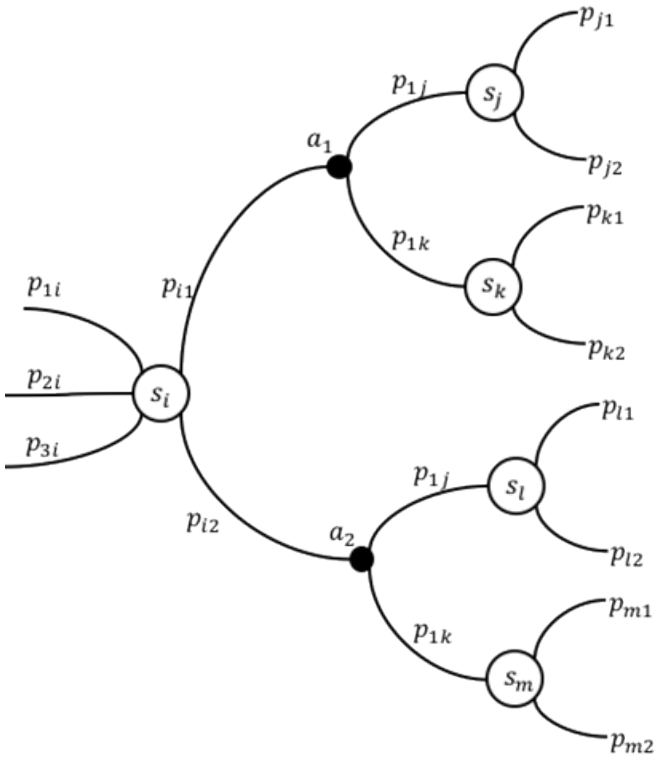


Figure 4.5: An illustrative example of a Markov Decision Process

The process of sequential decision making involves the computation of the series of transitions to reach a defined goal. This sequence of transitions is termed as the policy (π) and it, mathematically, is the function representing a mapping from the state space, S, to the action space, A, i.e.,

$$\pi: S \rightarrow \mathcal{P}(A)$$

In solving a problem formulated as a Markov Decision Process, an optimum policy is sought. In the execution of the policy a series of states are realized, given as,

$$\pi[(s_1, a_1, s_2, a_2, \dots, s_N)] \rightarrow [r_1(s_1, a_1), r_2(s_2, a_2), \dots, r_N(s_N)$$

In order to enable comparisons between policies, the concept of *partial ordering* between the policy realizations is to be developed. Puterman [131] defined this partial ordering as the "transitive, reflective and antisymmetric relationship between the elements" of the realization set. The definition of utility [136] for the realization set provides a concept of total ordering for the policies enabling their comparison along a real-axis. Due to the stochastic nature of the policy, two of them are ordered based on their expected utility [136] such that a policy π_1 is preferred over another π_2 if and only if,

$$\mathrm{E}^{\pi_1}[\psi(R_1, \dots, R_N)] > \mathrm{E}^{\pi_2}[\psi(R_1, \dots, R_N)]$$

where ψ is the utility function and $R_i = r_i(s_i, a_i)$

Puterman [131] suggested the use of a linear additive utility model in the comparison of the policies to model a risk neutral decision maker indifferent to the timing of the reward. Having established the ability to compare policies, the task of finding the optimal policy reduces to that of identifying the polity that maximizes the expected utility. Thus the framework of Markov Decision Processes provides a mathematical framework for the formulation of the sequential decision making process.

4.1.3 Reinforcement Learning

The field of reinforcement learning [137] is one of the branches of machine learning whose behavior lies in between that of supervised learning and unsupervised learning. The framework of Markov decision processes provides formalization for the definition of a reinforcement learning problem. In the standard representation of the framework, an agent interacts with a black box environment while attempting to optimize

its policy based on the returned signals and observed states without having been given specific guidance as to the actions that are to be taken. The set of actions to optimize the policy are expected to be learnt through experience. This process is illustrated in Figure 4.6.

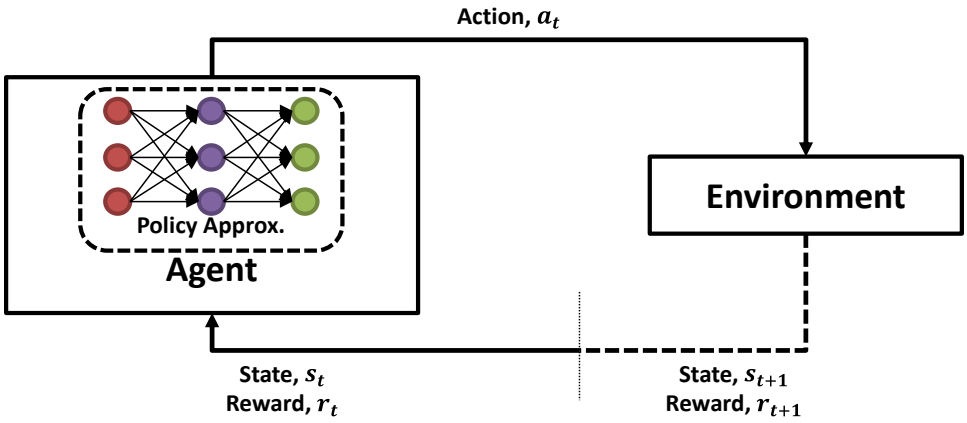


Figure 4.6: Standard definition of the reinforcement learning problem

The field of reinforcement learning has its roots in experimental psychology and the study of animal behavior, where the term "reinforcement" refers to the process of increasing the probability of reoccurrence of an event under certain conditions by rewarding the actions leading to the event [138]. This principle of learning from reinforcement is prevalent in engineering design, for example in the iterative design procedure, and motivated the earliest studies in artificial intelligence. The reinforcement learning framework all the treatment of partially-observable Markov decision processes along the same lines as that of the fully-observable Markov decision processes, granting it a distinct advantage over other methods, thereby enabling a unified framework all types of problems. The field of reinforcement learning finds its application in a vast variety of problems, for example, optimal control [139], [140], decision making in computer games [141], [142], resource management [143], robotics [48], [144], chemistry [145] etc.

The mathematical description of the reinforcement learning problem is slightly modified from that of the Markov decision process that was introduced in the preceding section through the introduction of a discount factor, γ, modifying the mathematical representation to $< S, A, T, R, \gamma >$. In the case of an infinite-horizon[4] problem, the total expected reward of the Markov decision process that was previous introduced tends to infinity when each individual reward is positive. To overcome this issue, the rewards are weighted as per their occurrence using the discount factor thereby keeping the total expected reward finite. This new policy utility, termed the "total expected discounted reward", is given by:

$$E^{\pi}[\psi(R_1, \ldots., R_N)] = \lim_{N \to \infty} E_s{}^{\pi} \left\{ \sum_{t=1}^{N} \gamma^{t-1} r_t(s_t, a_t) \right\}$$

for the case of infinite-horizon Markov decision processes, and for the finite case by,

$$E^{\pi}[\psi(R_1, \ldots., R_N)] = E_s{}^{\pi} \left\{ \sum_{t=1}^{N-1} \gamma^{t-1} r_t(s_t, a_t) + \gamma^{N-1} r_N(s_N) \right\}$$

In the infinite-horizon case, inclusion of a $\gamma \, \epsilon \, [0, 1]$, ensures that the cumulative reward is finite when all the individual transition rewards are finite. That is,

$$|E^{\pi}[\psi(R_1, \ldots, R_N)]| = M < \infty$$

when,

$$\sup_{s \, \epsilon S} \sup_{a \, \epsilon A_s} |r(s, a)| = C < \infty$$

[4] An infinite-horizon Markov decision process is one which has no terminal state and continues indefinitely.

Exploration plays an important role in the field of reinforcement learning in finding the optimal policy. Implementations of the reinforcement learning methods have to effectively balance the exploitation-exploration problem in order to ensure sufficient portion of the state space has been visited. A typical approach to this problem is to employ a decaying ϵ-greedy policy [146] in which greedy actions are chosen based on a probability mass function, skewed towards the greedy action by a factor of $1 - \epsilon$, with the value of ϵ reducing at every timestep.

The reinforcement learning concept of learning through observations differs from the theory of both supervised and unsupervised learning. In supervised learning, the learner is presented with a dataset consisting of labelled data in which a relationship is sought between the input values X, which is usually a MxN matrix and the output values y which would be a Mx1 column vector of either real or discrete values. The relationship that is developed would be represented in the form of a function approximation given by,

$$y = f(X; \boldsymbol{\theta})$$

and the supervised learning framework relies on finding the appropriate values for the parameters θ. As $f(X; \boldsymbol{\theta})$ represents an approximation of the true relationship between the inputs X and the outputs y, the parameters $\boldsymbol{\theta}$ are typically computed by minimizing a loss function representing the deviation of the estimated value from the true value. The regression loss is defined as:

$$Loss\big(y, f(X;\ \boldsymbol{\theta})\big) = \ \|y - f(X; \boldsymbol{\theta})\|_2^2$$

In most practical applications, a regularization loss is added to the loss term to protect the model from over-fitting the input data. The primary objective in supervised

learning is to generate a generalization of an unknown model in order to accurately predict model's behavior in previously unseen regions. The reinforcement learning problem on the other hand attempts to learn in an interactive setting, i.e., it is characterized by a dynamic environment, one where supervised learning is often impractical [137].

In unsupervised learning, the learning algorithm attempts to discover statistical structures in the input data without external supervision [147]. When provided with an input dataset X of size MxN, unsupervised learning methods either attempt to extract features by exploiting statistical regularities or attempt to build statistical models of the data for the purpose of density estimation, In both cases, unsupervised learning methods attempt to find the appropriate values for a series of parameters $\boldsymbol{\theta}$ which minimizes the degree of mismatch introduced due to the statistical modeling. In the case of Maximum Likelihood based density estimation methods, the degree of mismatch is given by the Kullback-Leibler divergence [148],

$$KL[\mathcal{P}_I(X), \mathcal{P}(X;\boldsymbol{\theta})] = \sum_X \mathcal{P}_I(X) \log \frac{\mathcal{P}_I(X)}{\mathcal{P}(X;\boldsymbol{\theta})}$$

As the primary objective of unsupervised learning is to identify patterns in the input data, it too differs from the formulation of the reinforcement learning problem which seeks to optimize the agent's policy while maximizing the observed cumulative reward.

4.1.4 Hypothesis 1

These differences in the theory of reinforcement learning, supervised learning and unsupervised learning necessitate the development of methods designed to enable learning in the reinforcement learning setup. Thus the framework of reinforcement learning provides the necessary computational tools necessary for the realization of the mathematical formulation of the Markov decision process. This leads to the formalization of the first hypothesis addressing the first research question, which deals with the application of the techniques of reinforcement learning for the purpose of automation of engineering decisions in a complex decision environment such as a design application. It is hypothesized that existing techniques and algorithms in the field of reinforcement learning can be utilized to automate the decision made in the engineering domain, and replicate human-level performance as observed in other domains.

Hypothesis 1
The framework of reinforcement learning provides the necessary computational tools that enable the replication of human-like behavior by artificial agents in the field of engineering design. This is enabled through the application of the mathematical tools offered by the Markov decision processes which form the mathematical foundations for reinforcement learning.

4.2 Research Area 2: Knowledge Extraction and Representation

As indicated in the previous chapter, the second research area deals with the identification of entities that need to be defined in order to enable the process of data-

driven knowledge utilization. While the current section does not introduce a concrete methodology for the knowledge extraction it does identify and define the necessary entities that have to be represented in order to enable learning. The second research area and the associated hypothesis build on the findings of the first hypothesis, i.e., there is an assumption made that the underlying framework that drives the automated learning is developed using concepts offered by the framework of reinforcement learning.

The parameters of the reinforcement learning problem that are associated with the knowledge stored in the system are that of the state of the system, the action taken by an engineer and finally the reward observed by the engineer that drives the next set of decisions. In the context of design applications, the state of the system or the environment can be represented by either the state of the design application or the state of the active design that is contained in the application. The extraction of the knowledge contained within the application can be accomplished through the utilization of a pair of infinite tails, one that tails the state of the application and another that tails any external resources created by the application. These proposed tails are infinite threads that are embedded within the design application and communicate with an external framework passing information from within the application in an encoded manner. This ensures that any interactive decisions taken by an engineer can readily be captured by the framework.

4.2.1 State representation

The hypothesis is associated with the extraction of knowledge associated with the state of the design from the design application is as follows. There exists an n-dimensional vector embedding space to which all the designs that can be represented by the design application can be projected. In the case of the UAV, the characteristics of the

UAV readily permit the mapping every design in from the application to such a space. In the case of a Siemens NX, one would need to develop a framework which accomplishes the task of projection of designs onto the manifold. In this n-dimensional embedding space, also termed as the manifold, each point would correspond to one state of a design. The design could either be realizable or not. For example, the combinations of components on the UAV that results in an incomplete representation of the system would be representative of an unrealizable state of the system. The above observations can be formalized into to the following theorem,

Given any universe, $\mathcal{U}$, that represents the possible representations of the designs within a design application, the state vector associated with any design within the application can be represented as a subset of the union of the states of all the alternatives of the universe. This universe of possible design representations occupy an infinite dimensional vector which can then be projected onto a finite dimensional embedding representing the state of the system at any instant of time.

Mathematically, this can be stated as follows:

Let $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}_2$, …, $\mathcal{S}_n$be the possible representations of the design alternatives in any universe, $\mathcal{U}$ such that, $\mathcal{S}_i \epsilon \mathbb{R}^\infty$, then for any new alternative that belongs to the architecture $\mathcal{S}_k$, we can write,

$$\mathcal{S}_k \subset \mathcal{U}, \text{ and } \mathcal{S}_k \epsilon \mathbb{R}^\infty$$

$$\text{as, } \mathcal{U} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_n$$

Further, as the hypothesis argues that there exists a lower dimensional embedding for each design in represented in the infinite dimensional space, we can generate formulate a mathematical problem that deals with the projection of this infinite dimensional vector onto a known n-dimensional embedding space where the state of the system is represented by $\mathcal{S}'$. This process of projection can mathematically be represented as,

$$\mathcal{S}_k \in \mathbb{R}^{\infty} \rightarrow S'_k \in \mathbb{R}^n$$

4.2.2 Action representation

The hypothesis associated with the extraction and representation of the action from the design space is very much dependent on the representation scheme chosen for the state of the system. In the context of a design application, the function of an action is to change the representation of the design application or the design contained within the application. Thus, from the perspective of the designs that exist in the proverbial infinite dimensional space, any action performed in the design application transforms an existing design in this infinite dimensional space to another through a sequence of infinite dimensional point traversals. In the context of the UAV, this can be represented by the change in any of the attributes associated with the UAV or in the context of the CAD model, this can be represented by the change in either the parameters associated with the model or a change in the topology of the CAD model. Thus, in summary, the action that occurs as a result of human decisions can be represented as the difference between the states of the design application or the product in the infinite dimensional space. This can be formalized as follows,

Given two states of a system, $\boldsymbol{\mathcal{S}_A}$ and $\boldsymbol{\mathcal{S}_B}$, in the universe,$\boldsymbol{\mathcal{U}}$, the effect of any action, $\boldsymbol{\mathcal{A}}$, transforming the state of the system from $\boldsymbol{A \rightarrow B}$ can be represented as the difference in the states of the states at $\boldsymbol{A}$ and $\boldsymbol{B}$.This representation of the difference of the states generates an infinite dimensional vector that can then be projected onto a lower dimensional embedding space.

which can be expressed mathematically as follows,

Let $\mathcal{S}_A$, $\mathcal{S}_B$ be two possible system states in an universe, $\mathcal{U}$, such that, $\mathcal{S}_{A,B} \in \mathbb{R}^{\infty}$, then, we define an action, $\mathcal{A}$, as,

$$\mathcal{A}: \mathcal{S}_A \rightarrow \mathcal{S}_B$$

and in general, we can define a policy as the set of all actions that results in the state transformation $A \rightarrow B$.

$$\pi = \{\mathcal{A}\}: \mathcal{S}_A \rightarrow \mathcal{S}_B$$

This implies given any two states, $\mathcal{S}_i$ and $\mathcal{S}_j$, it is possible to construct a policy,

$$\pi(i,j) = \{\mathcal{A}_{i,j'}, \mathcal{A}_{i,j''}, \ \ldots, \mathcal{A}_{i'',j}, \mathcal{A}_{i',j}\}$$

where each element of the set $\pi(i,j)$ represents a change in one single state variable, $\mathcal{S}_k[l]$. Further, as with the projection of the state vector to the lower dimensional embedding, it is hypothesized that it is possible to generation an embedded representation of the action in the lower dimensional space. This can once more be mathematically represented as,

$$\mathcal{A}' : \mathcal{S}'_A \rightarrow \mathcal{S}'_B \in \mathbb{R}^n \text{ and } \pi'(i,j) = \{\mathcal{A}'_{i,j'}, \mathcal{A}'_{i,j''}, \dots, \mathcal{A}'_{i'',j}, \mathcal{A}'_{i',j}\}$$

$$\text{such that, } \mathcal{A}_{l,k} \in \mathbb{R}^n$$

4.2.3 Reward representation

The final consideration that is to be made is the scheme utilized to represent the rewards associated with the system. In an engineering setting, each design represented by the design application has some indication of value associated with it. Engineers utilize this value as a representation of the goodness of a design that dictates their bias toward altering the state of the design through an action, i.e., the value associated with the state indicates the likelihood that the engineer would transition away from the design through by exercising an action. Thus, this implies that the reward associated with a transition can be expressed as the difference in the values associated with the states of the systems. In the context of the UAV, this can be represented as the difference between the key performance indicators associated with the UAV design such as the endurance, the range and the cost of the vehicle and in the context of the CAD model, the reward associated with a transition can be represented as the ability to get closer to a design that meets the design requirements imposed on the system. Thus, this can be formalized into a theorem stated as follows,

Given two states of a system, $\boldsymbol{\mathcal{S}_A}$ and $\boldsymbol{\mathcal{S}_B}$, in the universe, $\boldsymbol{\mathcal{U}}$, and an action, $\boldsymbol{\mathcal{A}}$, transforming the state of the system from $\boldsymbol{A \rightarrow B}$, the reward, $\boldsymbol{r(\mathcal{S}_A, \mathcal{A})}$, associated with the transformation can be formulated as perceived benefit of the transformation.

The perceived benefit of a transformation always resides in a finite dimensional space as the key performance indicators associated with the design are finite in number. In order to utilize this formulation in a traditional reinforcement learning setup, the multi-criteria reward formulation has to be transformed to a single source of reward and this can be achieved through the projection of the reward indicator onto the real-line. This hypothesis can be mathematically formulated as follows,

In a multi-criteria analysis, where the value associated with the state of a system, $\mathcal{S}_i$, is represented as,

$$\mathcal{V}(\mathcal{S}_i) \in \mathbb{R}^k$$

the reward associated with a transformation, $\pi = \{\mathcal{A}\}: \mathcal{S}_A \rightarrow \mathcal{S}_B$, can be formulated as the difference in the value of each successive pair of states along the policy.

$$r(\mathcal{S}_i, \mathcal{A}_j) = \mathcal{V}(\mathcal{S}_j) - \mathcal{V}(\mathcal{S}_i) \in \mathbb{R}^k$$

This reward can then be transformed to a single real-valued metric by projecting the k-dimensional reward metric to the real-line. This is given through the transformation,

$$r(\mathcal{S}_i, \mathcal{A}_j) \in \mathbb{R}^k \rightarrow r'(\mathcal{S}_i, \mathcal{A}_j) \in \mathbb{R} \approx r'(\mathcal{S}_i', \mathcal{A}_j') \in \mathbb{R}$$

4.2.4 Knowledge Representation: Knowledge Graphs

Having identified the sources of knowledge that have to be extracted and the mathematical meaning of the knowledge extracted from the design application, it is necessary to identify a means for the representation of the knowledge. The most natural representation of the knowledge is through the utilization of a knowledge graph, i.e., a

graphical model [149]. As the graphical model stores a tuple of node, edge and the consequent node as its underlying representation, this translates directly to the domain of reinforcement learning where the necessary entities to enable learning are the state, action, next state and the associated reward. This translates directly to the node and edge notation with the state of the system being represented as a node in the graph and the action being represented as the edge between two states of the system. As every state-action pair is associated with a deterministic reward, in terms of the knowledge graph, the reward observed within the system can be represented as an attribute of the edge of the system.

4.2.5 Hypothesis 2

The preceding passages highlight the proposed approaches for the extraction and interpretation of the various entities that are associated with the representation of the design application for the purpose of reinforcement learning, culminating with a proposal for the use of graphical models for the storage of data gathered from the data extraction routines. The observations made in the previous sections can be formally restated in terms of the following hypothesis.

Hypothesis 2
The reinforcement learning framework necessitates the extraction of three quantities in order to enable learning, these are the state of the system, the actions associated with the decisions made and the rewards indicated by the state-action transitions. In the engineering context, it is hypothesized that a mathematical formulation represented by

a n-dimensional embedding of the various entities can be generated in a manner that can either be application specific or application agnostic. Having generated these representations, the utilization of a graphical model forms the ideal means for representation of the entities associated with the reinforcement learning problem setup.

4.3 Research Area 3: Data-driven Knowledge Utilization

The third research area deals with addressing the question of the utilization of the knowledge without the involvement of hand-crafted heuristic. Existing research in the field of reinforcement learning draw parallels to the means in which humans learn from their decisions through the use of hippocampal replay as studied in the field of neuroscience [150]–[152]. The use of experience replay [153] is a well-established paradigm in the field of reinforcement learning that attempt to mimic this behavior of hippocampal replay of mammals. Recent studies in deep reinforcement learning have introduced the concept of prioritized experience replay [154] to overcome some of the issues associated with the biases introduced by the traditional experience replay learning framework. But none of the existing methods permits the consideration of experience replay in the presence of multiple sources of data. The established approaches to this problem combine the different sources of data into a single database from which the learning algorithm samples in order to update the agent's policies. This process is evident in the established demonstration based deep reinforcement learning algorithms such as the Deep Q-Learning from Demonstrations (DQfD) [155] or Deep Deterministic Policy Gradients from Demonstrations [156]. Thus in order to account for multiple different sources of data, the research work hypothesizes the utilization of a new sampling technique that extends the capabilities of prioritized experience replay by accounting for

data generated from different sources so as to bias the learning algorithm towards one of the source. This is achieved through the introduction of a separated buffer to store the data generated by the human demonstrator and that generated by the agent through experience. The resultant probability of the sampling is weighted based on the weighting associated with the source of the sample. The mathematical formulation for this sampling probability is given as follows:

$$P(d_i) = \begin{cases} \left(\frac{\delta_i + \epsilon}{\sum_k \delta_k + \epsilon}\right)^{\alpha}, & if\ d_i \epsilon\ Experience\ Buffer \\ max\left(\left(\frac{\delta_i + \epsilon}{\sum_k \delta_k + \epsilon}\right)^{\alpha}, \left(\frac{h_i}{\sum_k h_k}\right)^{\beta}\right), & if d_i \epsilon\ Demonstration\ Buffer \end{cases}$$

where, δ represents the temporal difference error given as,

$$\delta_i = r\left(\mathcal{S}_A, \mathcal{A}_{A,B}\right) + \mathcal{V}(\mathcal{S}_B) - \mathcal{V}(\mathcal{S}_A)$$

$$\beta_i \propto \frac{k}{\delta_i}$$

with the terms r representing the reward observed as a result of the transition from state $\mathcal{S}_A$ to $\mathcal{S}_B$ and the value of the state $\mathcal{S}_i$ is represented by the term $\mathcal{V}(\mathcal{S}_i)$. The terms α and β represent the annealed weighting applied to the different probabilities associated with the sampling routine. Finally, the term h_i represents the sampling weight associated with the source i. This formulation ensures that when the network is biased toward the source with the highest weight so as to ensure demonstrations from the best source are considered more frequently in the learning process. Additionally, the method would also ensure there isn't significant variance in the prediction of the state value as when the error associated

with the prediction increases the samples with greater error are preferred regardless of the source of the sample.

These observations can be formalized into the third and final hypothesis which can be stated as follows,

Hypothesis 3
With the extension of the framework of prioritized experience replay by modifying the computation of the priorities associated with the probability of sampling, it is possible to account for demonstration data from multiple different sources. Further, by altering the storage framework by separating out the experience and demonstration buffers, it is hypothesized that the agent would be able to leverage demonstrated data in learning the optimal policy.

CHAPTER 5. KNOWLEDGE-BASED LEARNING FRAMEWORK

The previous two chapters introduced the research questions, the associated hypothesis, the research methodology and the applications that are considered as proof-of-concept problems but before the methodology is applied to a particular problem, a generic framework for doing so is introduced in the current chapter. The framework that enables the application of the research methodology towards the automation of engineering decisions for design systems is identified and the architecture of the framework and the data-models associated with it are introduced in the current chapter. The chapter is structured as follows, first, a set of requirements for the framework are identified and classified that dictate the functionalities that are to be included in the application. This is followed by the identification of off-the-shelf software components that enable the realization of the requirements. These are finally combined together for the realization of the architecture of the framework.

5.1 Guiding Requirements of the Framework

The development of the knowledge-based learning framework is driven by the identification of a set of engineering and user requirements that dictate the necessary components or features that have to be included in the final product. In the course of definition of the requirements, it is assumed that the resultant framework would meet all the prerequisites set by the research goal, i.e., the framework would be independent of the design application, the framework would be capable of extracting knowledge from the application in an automated manner as a user interacts with the it and that the framework would utilize the concepts of reinforcement learning to enable knowledge-based decision

making. Thus, through the established research questions and their corresponding hypotheses, a set of four key requirements groups are identified, illustrated in Figure 5.1. These are,

- System engineering requirements
- Reinforcement learning requirements
- User experience requirements
- Requirements on the computational capabilities

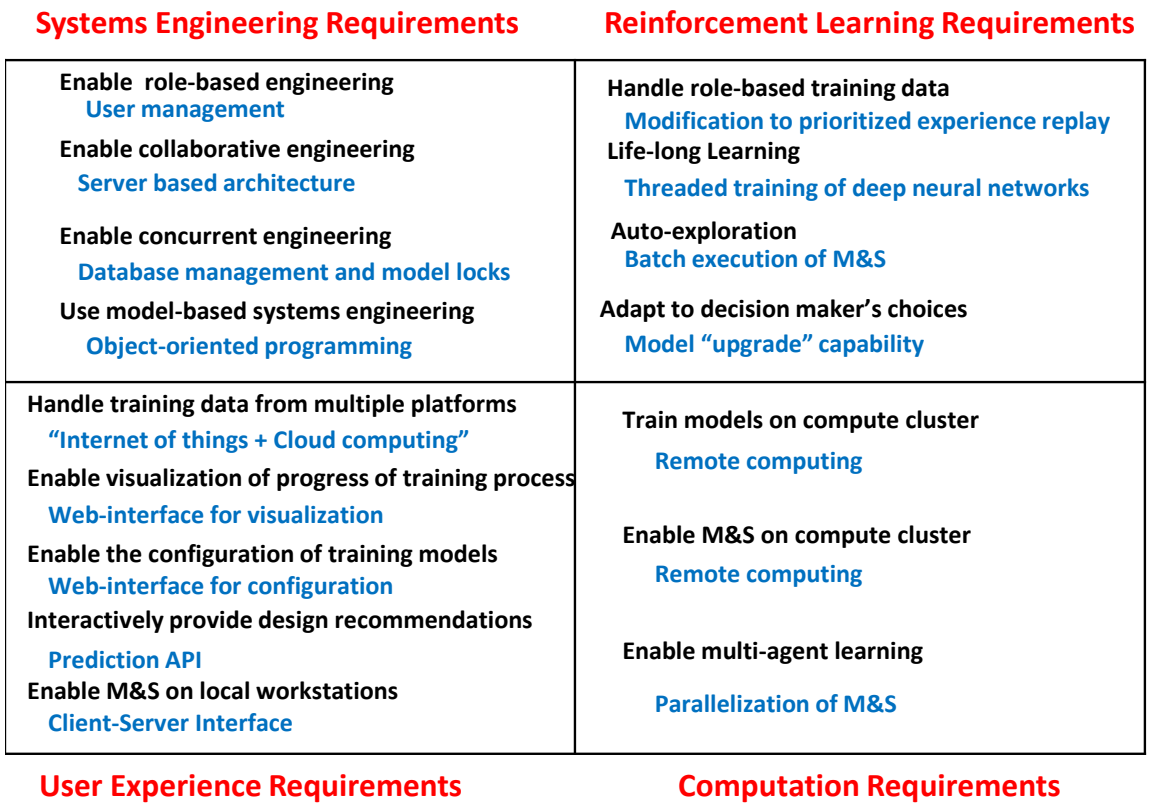


Figure 5.1: Desired requirements imposed on the framework in order to guide the development process

5.1.1 System Engineering Requirements

The systems engineering requirements function as the bridge between the reinforcement learning framework and the design applications. They guarantee that the application to which the framework is applied meets the requirements of parameter, design and value definitions identified by the second research question. Further the implementation of a systems engineering capable framework would enable a process of automated verification and validation, i.e., design evaluation, which in turn acts as an indicator of a goodness of a design and hence the associated design decisions. In the current dissertation this is accomplished by the development of a model-based systems engineering (MBSE) framework. A MBSE framework with its inherent object-oriented representation of the structure of the designed product is aligned with the frame-based data representation scheme hypothesised in CHAPTER 4. The details of the model-based system engineering framework are addressed in CHAPTER 6 in the context of the problem of design of unmanned aerial vehicles. Additional requirements that arise from the consideration of the manner in which the design applications generate data are that of the source of the data and the frequency of production of data. As in a typical engineering setting, data can be generated by multiple engineers there may arise a necessity to distinguish between the value of each data point. For example, if a certain problem data is generated by an engineer considered to be an expert in the field, such data points would contribute more significantly to the efficient learning of an algorithm in comparison to that generated by a novice engineer. This can, in turn, be phrased such that the framework being capable of handling user roles and thus making a distinction between the data generated by users of different roles. In the current dissertation, two roles of engineers are considered: the expert and the novice engineer. In an engineering setting, a typical

problem is addressed by a set of engineering teams who may be spread across different parts of the world. As the data generated by each team would can be viewed as training the same learning system, there is a necessity for the framework to handle data from multiple different sources, i.e., different instances of the design application. This can be viewed as enabling the process of collaborative engineering. Finally, the nature of engineering design is such that different teams work on different portions of the design problem in parallel. This leads to the parallel generation of engineering data that, while related, are often associated with different design decisions. In order to accommodate such a scenario, the framework has to able to handle the issue of concurrent generation of design knowledge.

5.1.2 Reinforcement Learning Requirements

The requirements placed by the reinforcement learning framework involve the manner in which an artificial agent would learn from the extracted knowledge. The primary requirement comes from the fact that the generation of data generated would occur from multiple sources, i.e., expert and novice engineers. In such a scenario, there is a necessity for the modification of established learning mechanism to account for the contributions of multiple sources of data. Another consideration to be made is that of the manner in which an agent would traditionally learn. In the traditional reinforcement learning setting, an agent is trained against a static prepopulated database for a certain duration or until the agent fails to improve. But in the current setting, given that the knowledge base is dynamic in nature, it is necessary to consider a dynamic training setting for the agent. To this end, it is proposed to leverage life-long learning [157] and transfer learning [158] where agents are constantly trained with changes to the knowledge

base. Thus, it is necessary for the framework to enable a means for the dynamic adaptation of the agent during its utilization process. Finally, in order to ensure the identification of the best possible set of decisions, often learning from purely the set of demonstrated data may be insufficient. This may be a result of the biases in humans that prevent them from making the best possible decision at every step or due to the nature of the problem where the human operators are unaware of the best possible decisions to make. In such a case, it is necessary for the artificial agent to be able to gather knowledge by auto-exploration. This of course restricts the design applications to which the framework can be applied as in order to enable auto-exploration, the design application has to provide a capability for the batch-execution of the modelling and simulation process.

5.1.3 User Experience Requirements

The user experience considerations for the development of the framework arise from two sources, the first form the manner in which the users would interact with the framework to develop an agent's model and the other for the manner in which the data for the training of the agent is generated. In the first case, the configuration of the agent's models is to occur at a design *project* level. Given that multiple teams across different locations work on the design problem, it is proposed to centralize the location for the network definition and configuration. This also enables the framework to work with data from different sources of knowledge, i.e., knowledge being created from not just personal computers but also other devices. This, of course, necessitates the of standardization of the interface between the framework and the design application. Finally, it is desired that the framework interface with the design application to provide recommendations to the

user in an interactive manner as and when requested by the engineer. This implies that the framework has to be capable of calling upon the trained agents to make predictions regarding the actions that have to be taken by the engineer using the design application. The means of representation of the recommended decision is assumed to be handled by the application.

5.1.4 Computational Requirements

The requirements of the computational capabilities arise from the training durations that may be required to adequately train the agent. Due to large amounts of computation required, there is a necessity to trigger the learning process on compute clusters capable of providing the necessary computation capabilities. Further, in the process of auto-exploration, there would be a necessity to execute the design application on these compute clusters so as to enable the generation of exploration data. Finally, there may be the necessity to train multiple agents as in the case where decisions associated with different disciplines requiring an agent to represent each discipline of the design process. In the absence of any interactions between the decisions and the underlying disciplines, it is possible to train the agents associated with the decision in parallel. Thus, the framework needs to provide a capability for both the parallelization of the training process and the parallel training of multiple agents.

5.2 Framework Architecture

Based on the requirements posed, a server-based framework architecture is developed. The framework would necessitate the inclusion of a knowledge base, a retrieval mechanism for the knowledge and a standardized interface between the server and the design application. In order to streamline the flow of information, two separate

servers are setup by the framework; one to host the database and another that serves as the computation and training server. Owing to the growing popularity of the Python programming [159] language [160], its open-source availability, its interpreted nature and the ability for quick prototyping that is intrinsic in the language, the backend of the framework is developed in Python. Further, open source packages developed by the Python community and free software alternatives are leveraged in assembling the components of the framework in order to ensure reproducibility of the framework demonstrated in the dissertation. In the development of the knowledge-based reinforcement learning framework, there are two levels of abstraction that are created. The first of the two handles the project, user, and item management tasks while the second addresses the problem of learning. These layers are stacked on top of each other creating the final framework enabling knowledge-based learning. The following passages address the components of the framework and highlight the interactions that occur between them and the design applications, and the following section highlights the data models and their interactions that constitute the framework.

5.2.1 Framework Components

The architecture of the framework is highlighted in Figure 5.2. The figure clearly identifies the two constituents blocks of the framework, the database server and the computation server. The architecture for the database server builds off of the object-oriented MBSE framework proposed by Balestrini-Robinson et. al. [161] in which a database server is configured with NoSQL and graph databases to store application data supplied by the backend of the sever-based application forming the core element of the framework.

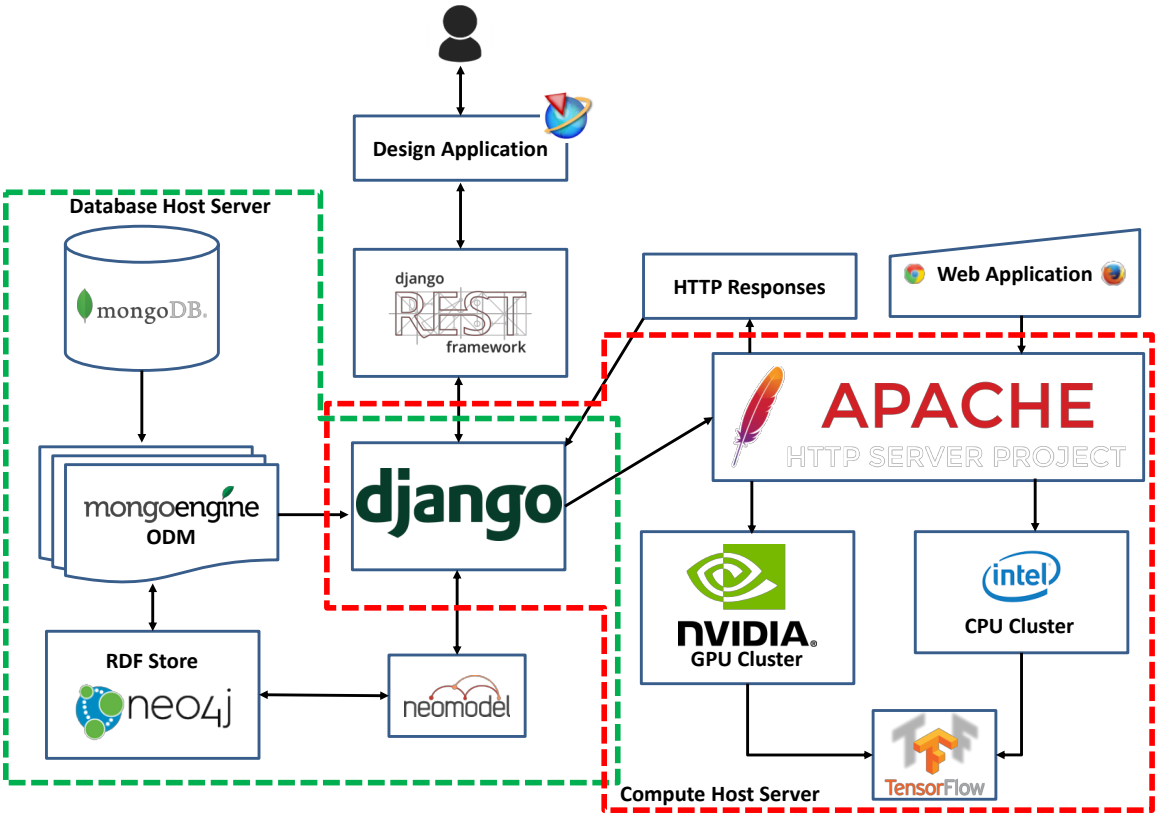


Figure 5.2: Architecture of the framework that enables knowledge-based learning and automation

On the other end a computational server comprised of a set of graphical processing units and central processing units are assembled to represent the computation element of the framework. These, too, interact with the backend of the framework where learning data queried from the database and signals indicating the learning status are passed to the learning algorithms to control the nature of training.

5.2.1.1 Databases

NoSQL database stores all the data that is utilized in the framework to aid the process of learning. The data generated could correspond to the instance of the design application from which knowledge is created, the actual knowledge generated by user

actions, and also the models that are generated as a result of the learning process. The choice of a NoSQL database stems from the ability of such databases to store ASCII encoded data in a text-blob means. This enables the storage of complex relations between the data models while preventing the necessity for the definition of explicit relations between that characterize standard tabular or relational databases. The NoSQL database is realized through the use of free NoSQL database provider MongoDB [162]. In order to ease the communication between the framework which lives in the Python session creating data models and learning models as objects and the database that stores these models as documents, an object-document mapping package, mongoengine [163], developed in Python is utilized. An inherent drawback of utilizing NoSQL databases is its inability to handle complex retrieval queries, such as perform complex logical operations between elements of the stored dataset. In order to address such an issue, in parallel to the NoSQL database, a graph database is utilized where information associated with the objects created in the NoSQL database are stored in the form of relational triplets with the edge identifying the relation and the nodes characterized by the unique identifiers associated with the model objects. As highlighted by researchers [164] RDF databases in contrast to NoSQL databases are capable of efficiently handling highly complex queries. This is leveraged and an RDF store database in the form of a graph database is utilized. This is represented by the “neo4j” node in the architecture of the system. Neo4j is chosen as it not only provides a community version of the graph database free of charge, but also due to the availability of bindings to the graph framework in the Python programming language, such as neomodel, Neo4jRestClient etc. Thus, in the current dissertation the open source Python package neomodel [165] is utilized as the interface between the backend of the framework and the graph database, neo4j [166].

5.2.1.2 Computational Setup

The other part of the framework is represented by the computational server that comprises of a set of computational nodes that are configured to train the learning models. The learning models are trained using the open-source dataflow programming library, tensorflow [167]. In order to enable memory intensive computation, there are two compute nodes utilized. Each node is comprised of two nVidia Tesla 8GB K10 GPUs and a set of 16 Intel Core i7 processors. Two separate tensorflow configurations, one for the GPUs and another for the CPUs, are utilized to obtain optimized performance across each training task, although the training of an agent is restricted to one single GPU per node, while no such restrictions are placed on the CPU. As a result of the differences in performance of the GPU and CPU, where there was an marked (x30) improvement in the training speed as a result of the utilization of the GPU, the framework was configured in a manner such that all the model training occurred on the GPUs while the evaluation and prediction of the models was carried out on the CPUs. The compute server was built on top of an Apache HTTP web server [168], which provides an open-source cross platform web server that enables the communication between the framework and the design application.

5.2.1.3 Integration and Configuration Unit

The integration and communication between the computation and database server is carried out on a third, and final, host server. The choice of having a separate server is made in order to avoid unnecessary loads on either the computation and the database server. The integration is carried out through the utilization of the REST framework which provides a language neutral means of communication across machines. The REST

API is built off of a Python backend for the web server developed in the open source Python package Django [169], with the API being developed through the use of Django-Rest-Framework [170], a Python package that enables the development of streamlined web APIs. While a user interface may provide an easier experience in terms of the configuration of the models being trained, the current research work achieves this through the use of web APIs, with the development of a web-based interface put off for future work. Finally, the computation server hosts the visualization framework, tensorboard [171], that enables the visualization of the progress of the training process across all the agents that are trained by the framework. The integration unit and the host server also function as the interface between the design application and the framework through which data is imported into the databases and predictions are exported from the trained models.

5.2.2 Data Models

The current section addresses the data models that are used in the framework and the interactions between them that enables the development of an application agnostic knowledge-based learning framework. The primary abstraction of the data models occur at two levels, the first of which generates abstract representations for the design application, the design problem and the active design instance, while the second deals with the representation of the artificial agent, the mechanism for training and the knowledge extracted from the design application. Each design application is identified by an active *application instance*. The instance is registered on the framework when the design application is launched and is terminated when the application is shut down. In addition, it is assumed that each design application can work on one single *project*

instance at any instant of time. The project is used as a representation for a design process or design activity. The result of the project is the creation of a set of *items* that are the abstract representation of the result of a completed design process. Finally, each project is associated with a set of *users* that manages the access for the users in the project and this is further refined by access criterion placed on the items. The UML diagrams for each of these models is shown in Figure 5.3 where the contents of each of the data model is highlighted.

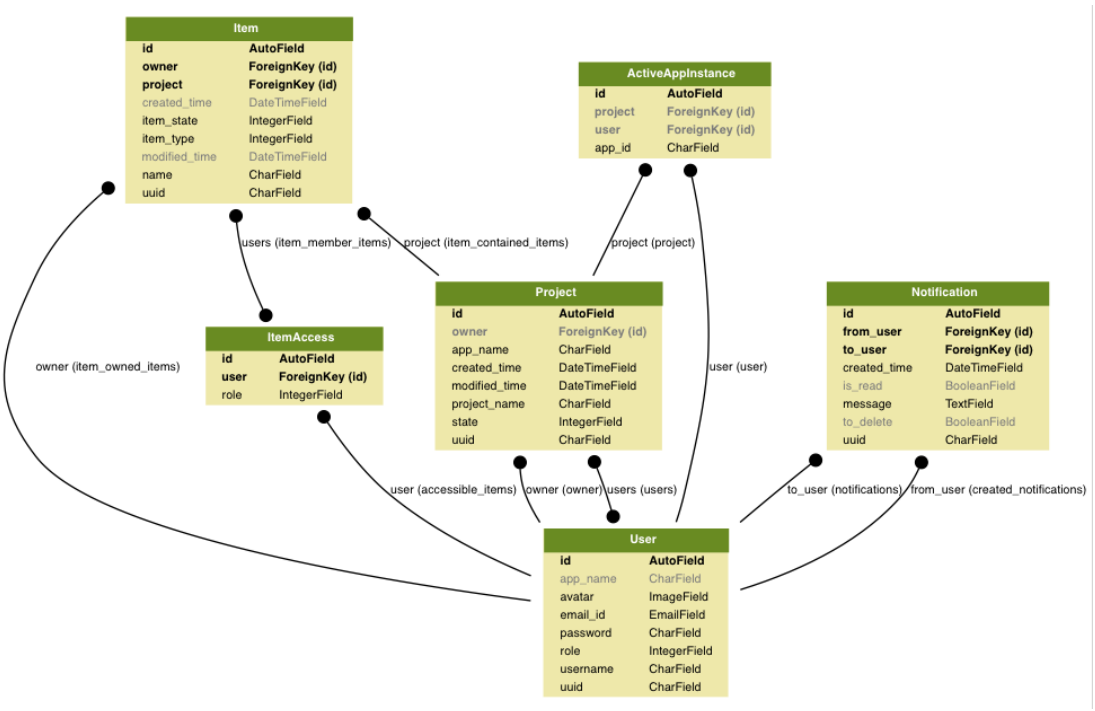


Figure 5.3: Application level data models contained in the framework

The key feature to note among these data models is that the project in the framework can exist in three states, in preparation, completed and obsolete. Projects in preparation are only visible to engineers who are members of the project, released projects are one that are visible to all engineers in a certain group and the released status is assigned when the project has been completely setup and the analyses are either ready

to be or have been executed. In order to enable collaboration between users, the framework implements a notification service that enables the transmission of messages and event notifications between users. In order to ensure concurrent engineering can be implemented, the database nature of the models used to represent the designs, ensures consistency in the data stored in the database thereby preventing conflicting edits to common aspects of the problem. This is achieved by the association of a *state* representing the edit state of an item at any instant of time. When the state of the item is altered by a user to represent the edit operation undertaken by that user, the attributes of an item upon commit are locked from being edited by others thus preventing conflicting edits to the same attributes. These changes are then notified across all other users who have the item active in their corresponding design applications. The concept of notifications is also utilized to enable communication within the framework where signals are passed from the knowledge database to the computation server to activate the agent's learning process or update an agent that is used as the predictive model. The connectors that have a single symbol at its end indicates a composition, while the connector having a symbol on either end represents a many-to-many relationship. While the database used NoSQL, a semi-structured relationship definition is utilized in order to standardize the nature of interactions permitted with the framework and also enable a better documentation of the framework for future development efforts.

On the other end of the framework are the data models that are associated with the learning models, the training model configurations, the associated algorithms and the knowledge that is stored within the database. These models are illustrated in Figure 5.4 and revolve around the creation of a *ControlItem*.

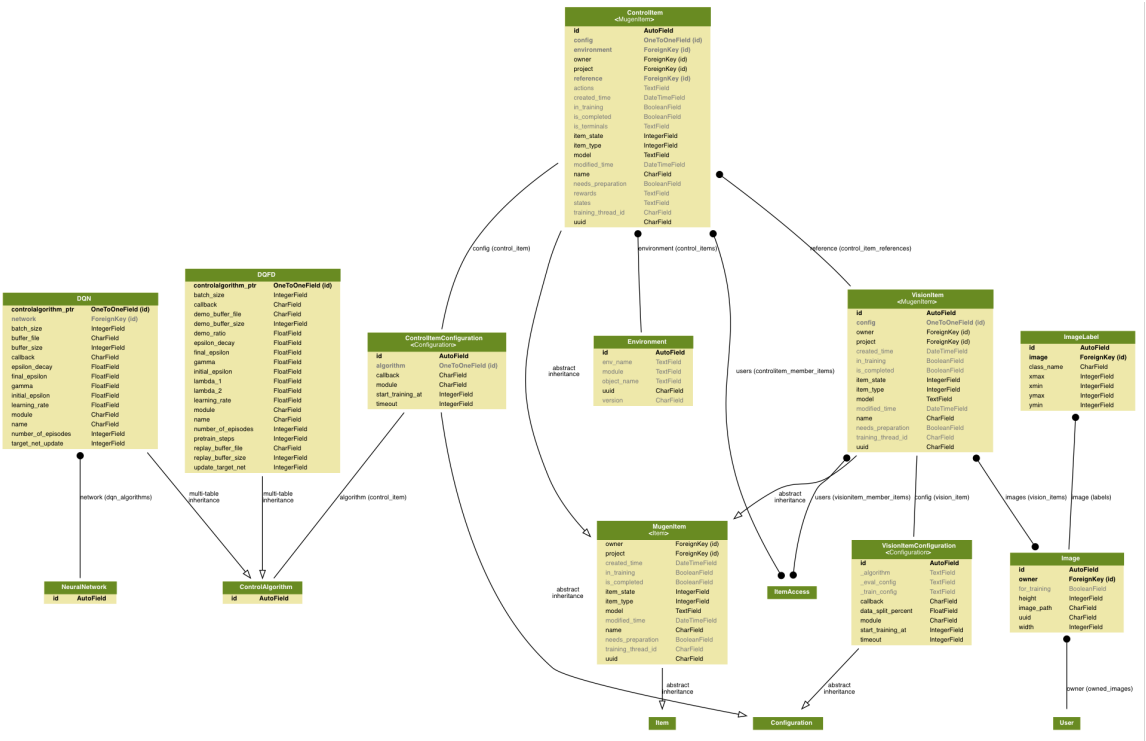


Figure 5.4: Data models associated with the creation of the artificial agent

These control items function as containers for the agent containing not only the data to be utilized in the training process, but also the configuration for the training algorithm and the agent's model. The control item is configured through the specification of the design application that it is to be associated with, so as to enable any resultant agent automated control of the design application during the auto-exploration phase. In addition to the specification of the *environment*, a set of *configuration parameters* controlling the nature of the learning process are to be specified during the creation of the control item. The control item also supports the specification of the *training mechanism* or algorithm, which in turn can be configured based on the type of algorithm chosen. Each algorithm would support one or more *agent model* would need specification in order to completely configure the control item. Having specified all the configuration options, the training process would await the number of training samples specified in the

options to start the agent's training process. Each training sample would be represented by a *data point* that would comprise of a *data source*, i.e., user creating the data point, *state* of the design application or the design model, the *action* taken by the user, the *resultant state* of the system and the *observed reward* as a result of the transition. While not implemented in any of the use cases, the framework supports the ability to model the state of the system as an image enabling a visual perception of the design application. The extension for the inclusion of visual perception is planned to be addressed as a future work of the current research. Figure 5.4 also illustrates the relationships that exist between the data models of the artificial agent. In addition to the relationships, the figure also indicates the aggregation and inheritance in the data models displaying how different models can be utilized to realize a knowledge-based learning framework.

5.3 Event sequence in the framework

There are two modes of events that the framework is comprised of, the first being the knowledge extraction event and the other the knowledge utilization event. In the knowledge extraction mode of operation, an engineer's decisions are automatically identified in order to extract knowledge contained in those decisions and represent these in an appropriate means for the training of the artificial agent. The knowledge utilization mode relies on the presence of a trained agent that can supply recommendations as to the actions that are to be taken by an engineer when faced with a design problem in the operation of the design application. The modes of operations are illustrated in the Figure 5.5 and Figure 5.6. The details of the operation of in the different modes are addressed in CHAPTER 7.

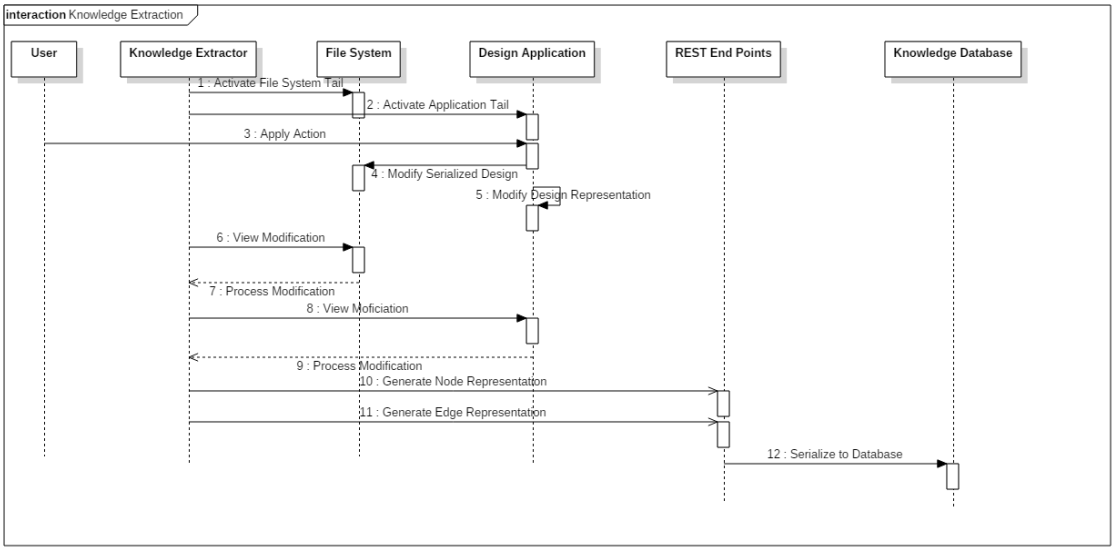


Figure 5.5: A high-level representation of the sequence of interactions that occur between the elements of the framework in the extraction of knowledge from a design application

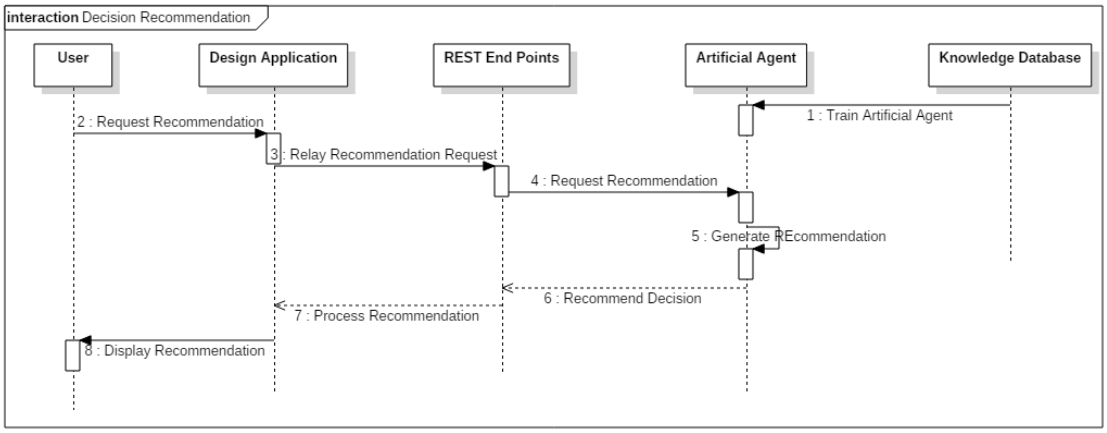


Figure 5.6: A high-level representation of the sequence of interactions that occur between the elements of the framework in the generation of a recommendation of a decision

CHAPTER 6. USE CASE I: MBSE APPLICATION FOR UAV DESIGN

The current chapter introduces one of the two use cases to which the framework is applied. The first use-case deals with the problem of design of an unmanned aerial vehicle using a Model-based Systems Engineering application. The purpose of the use case is to demonstrate the capability of the reinforcement learning framework to replicate human-level decision making in the presence of demonstrated data and to improve over the demonstrated performance through explorations. The use case also evaluates the feasibility for the transfer of knowledge from one scenario to another where there is a change in the problem formulation in terms of the requirements. In order to enable a system-engineering based formulation of the problem, the research work develops a Model-based Systems Engineering framework that enables the application of knowledge extraction and representation methods. The application is developed in PyQt [172] so as to simplify the knowledge extraction capabilities and to develop a prototype for the demonstration of the learning capabilities. In this use case, knowledge is extracted in an automatic means as opposed to being extracted to being demonstrated by humans. The consideration of incorporation of human-decisions in the training the artificial agent is made in the other use-case. The chapter addresses the following topics,

- Development of a MBSE framework for the purpose of knowledge extraction and representation.[5]

[5] The work done on the development of the MBSE framework has been published as part of the paper "On-Demand Small UAS Architecture Selection and Rapid Manufacturing Using a Model-based Systems Engineering Approach" at the ICAS 2018 conference **[Justin2018]**. The author of the dissertation is one of the two leading contributors to the published paper.

- Introduction of the mechanism for knowledge extraction and the associated encoding scheme for the implementation of the learning algorithm.
- The formulation of the learning problem, in terms of the metrics associated with the use-case and the means for incorporation of requirements into the formulation of the problem.
- Finally, a detailed analysis of the results observed from the execution of the learning algorithm in different scenarios, such as an exploratory algorithm without demonstrations, demonstrations from multiple sources, and finally, the ability to generalize across different combinations of components and requirements.

6.1 Model-based Systems Engineering Application

Model-based Systems Engineering (MBSE) [173] is a process in which models are used in the description of the product being design such that the resultant engineering process would enable the generation of a virtual engineering framework that accounts for all the different elements of the product design life cycle, ranging from requirements engineering to verification and validation of the designed product. Due to this capability of enabling a virtual representation of the design process and the feedback of results into the design through the virtual verification and validation process, the MBSE process can be utilized to generate an estimate of requirements satisfaction. This key concept is leveraged in the current work. Although several MBSE applications do exist in the market such as MagicDraw [174] or Enterprise Architect [175], most commercial applications do not meet the requirements posed by the research work. These system typically are restricted to being a modelling environment, without the capability for

simulations or completely black-box applications that do not provide access an API for the extraction of knowledge from the application. Thus, as an alternative to these applications, a simplified version of a MBSE application is developed in the current research work.

Traditionally, the MBSE approach builds on the model-centric view established by Model-Based Engineering (MBE) [176] by extending it to the field of systems engineering. The well-established model-centric view of engineering relies on the use of models as an integral-part of the baseline. With the recent surge in the demand of unmanned aerial vehicles there have been considerable investigations into the use of MBSE-based approaches for the design of UAV [177]–[180] While these past investigations into the development of MBSE-based approaches for the design of the UAV have looked into means for improving the design processes and the design cycle time from the perspective of automation, they do not address the issue of capture of design knowledge from the MBSE application. As such, the methods and processes proposed in by the researchers thus far rely on expert engineers setting up an automated process using which the automation of tasks can be accomplished. The underlying issue with these methods is the inability for the system to learn from the experiences observed. This is addressed in the current research work with the development of a simplified version of the MBSE application in which artificial agents extract knowledge from the design system while the design system is in use and provide recommendations to the users, when requested. The application is built in a manner to provide the artificial agent the capability of exploration of actions that constitute the design process.

6.1.1 Background

The framework of MBSE enables life-cycle management of a product through the representation of each of the element of the design as a model. This includes the generation of models for activities such as requirement analysis, performance analysis, design optimization, requirements verification and validation. Thus, the MBSE framework accounts for all the elements of the life cycle of the product, from the beginning of the conceptual design phase with the definition and handling of requirements, to detailed design and manufacturing, and to the later life-cycle phases of validation, operations support, and maintenance. Overall, an MBSE framework improves communications across development teams, reduces design cycle times, and reduces risks through the identifications of failures earlier in the design cycle [181]. It has to be noted that MBSE provides a framework through which systems-engineering based design can be carried out, but it does not provide a methodology or a language in which to carry out the design process. Based on the results of the comprehensive review of established MBSE methodologies [182] and languages [183], a choice to extend the methodologies and language capabilities offered by System Modeling Language (SysML) is made. The SysML provides a language that can be utilized to describe the contents of the product being designed through the extension of a subset of the Unified Modeling Language (UML) protocols. Figure 6.1 represents the key elements that form a part of every model defined using SysML and also illustrates the interactions that occur between these components.

The four elements identified by the figure are termed as the pillars of the SysML model and are described as follows,

- The requirements package describes the desired characteristics of the product being designed in terms of the product behavior, design and operation. These establish bounds and checks on the actual product behavior and the goal of each engineering process is the satisfaction of the defined set of requirements to a certain degree. By identifying the appropriate set of requirements in a design problem, it is possible to establish the set of objectives for the design problem along with its constraints.

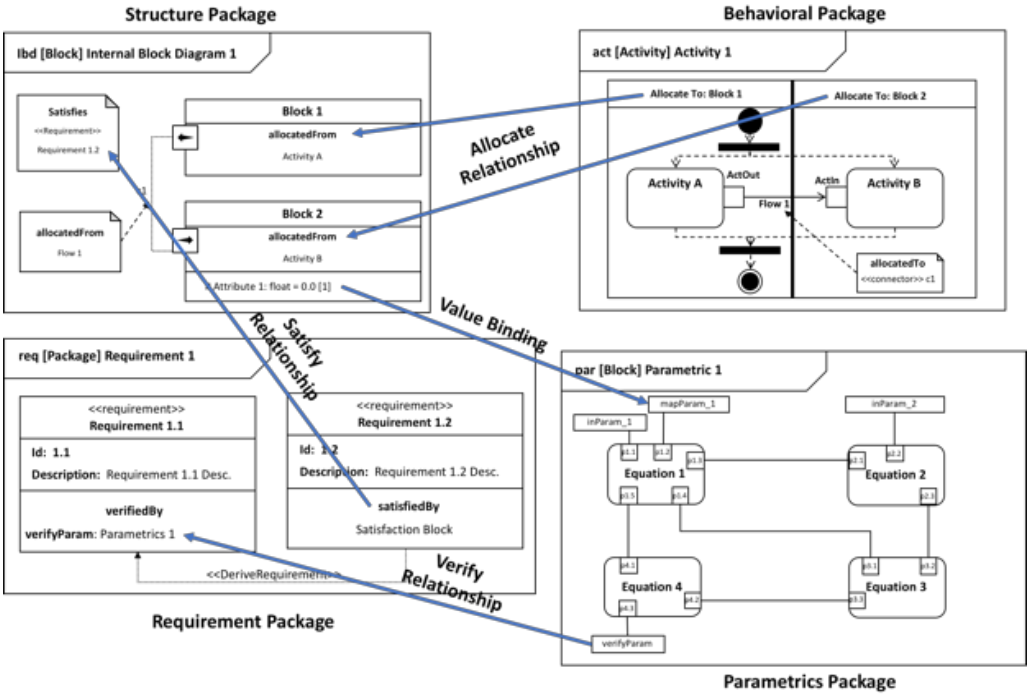


Figure 6.1: Elements of a model in SysML and the interactions that occur between these elements [181]

- The behavior package generates models for the functional and physical behavior of the system being modeled. The package describes the interactions that occur between the components of the system from a functional

perspective while providing an indication of changes that occur to the component when certain parameters are altered on the system.

- The structure package models the contents of the system being represented by the model. The structure package describes the constitution of the system in terms of the subsystems and the components and also describes the physical connections that exist between these subsystems. The usage of SysML further enables the specification of possible connections that can exist between the elements of the system.
- And finally the parametrics package which describes the relationships and bindings between different attributes defined in the elements of the structures package. The parametrics are typically used to represent the physics-based or statistical relationships that govern the manner in which the parameters associated with the elements in the system can vary.

As SysML is only the language that governs the specification of the model of the system, it is not inherently executable. This in turn results in the reliance on external applications to simulate the behavior of the system after having modelled the system being designed [177], [183]. Further, as designers may not be familiar with the fundamentals of systems engineering making the adaptation of the systems engineering paradigm of design challenging. The developed framework adapts the traditional view of SysML with two modifications. These are,

- In order to mimic the nature of complex engineering design, the framework merges the behavioral and parametrics package into one single “processes” package. Each process within this package is formulated as a design structure

matrix (DSM) [184] a representation that is an accepted standard for the representation of complex engineering processes in the engineering domain [185].

- Secondly, the data structure associated with the representation of the elements of each of the package is standardized. Every package represents the data comprised within it in the form of a tree-data structure, with interactions between elements within the trees being defined by a new type of standard attribute called the "interface".

The adaptation of the standardized representation and the introduction of an executable capability for the SysML framework eliminate the need for intermediate model converters in the execution of the design process, thereby bypassing the learning curves associated with the traditional SysML practices.

6.1.2 Modelling the UAV

The modeling of UAV per the specifications of MBSE requires the development of computational representation of the requirements, the system structure, the component interfaces, and the processes associated with the design and manufacturing activities. The following section details the development of the models necessary to analyze the capabilities of different UAV architectures, thereby evaluating their capability to satisfy the requirements posed in terms of the mission performance.

6.1.2.1 Requirements Modeling

The process of engineering design begins with the definition of a set of requirements that drives the consideration of the design process. The traditional Forsberg and Mooz Systems Engineering "Vee" model [186] relies on the presence of a decoupled

process of requirements definition and that of the requirements validation, even though, the process of requirements verification and validation forms an integral part of the process of automation of engineering decisions. Thus, having an executable MBSE framework in which requirements are automatically mapped to the metrics of interest alleviates the issues faced by the decoupled approach of requirements definition and decomposition and that of the requirements validation and the recomposition. The presence of such an environment would imply that as and when parameters are altered on the system, the evaluation of process models that reference these parameters can be automatically triggered and the outputs of these processes can be mapped back to the requirements to provide an indication of if the design meets set of requirements with an indication of the source of dissatisfaction, if any. From the development of the application, two prerequisites are imposed on the requirements package. First, it is assumed that each requirement defined by an engineer is stated in natural language. This implies that there would be a necessity for the application to process natural language and to convert this representation into that of engineering using appropriate metrics. Secondly, owning to the difficulty in translation of requirements to constraints and the mapping of requirements to physical entities in the designed structure, an automated framework is necessary to handle the association of requirement models with that of the define structural elements. This would again necessitate the ability for processing of natural language and engineering language.

To achieve these targets, the field of natural language processing (NLP) [187] and text parsing are used. The process of engineering design begins with the elicitation of requirements which typically results in the curating of requirements stated in natural language. In the case of a UAV, this can be viewed as being a specification of the

vehicle's desired functionality from which an engineer would derive the performance metrics to guide the design process. In the aircraft design community, one of the main sources of the requirements that triggers the design process of the aircraft is the mission profile from which the engineering metrics can be extracted. In order to explain the process involved in the extraction and identification let us consider an example of a mission requirement,

"The UAV shall fly a distance of 1,800 m within 2 min."

The methodology implemented that translates the description of the requirement into a verifiable model is four-fold and builds on the methodology demonstrated in the past [188][189]**.** The process is illustrated in the Figure 6.2.

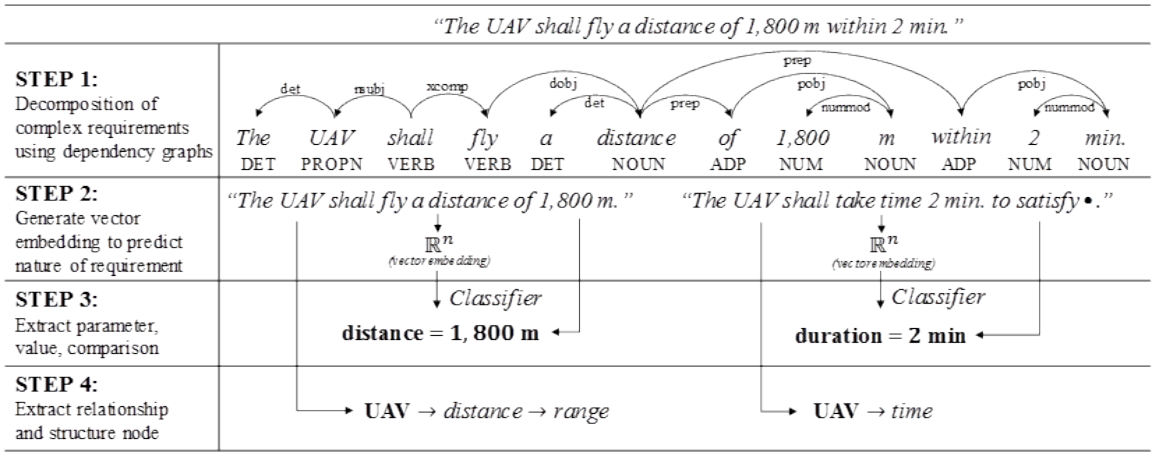


Figure 6.2: Algorithm for the processing of requirement described in natural language using NLP

- First, a dependency tree [190] is built so as to identify the elements of the sentence thereby breaking complex requirements into a set of simpler requirements. In the above scenario, given that there are two objects of prepositions, the input requirement can be decomposed into two, one that deals

with the range associated with the attribute "*distance*" and value "*1,800 m*" and another that deals with the duration associated with an implicit attribute "time" and the value "*2 min".*

- Second, once the requirement is decomposed into its constituents, a vector embedding [191] is generated for each individual requirement. This vector represents the projection of the natural language requirement into an n-dimensional real space where similar requirements occupy a common region of the space. This is exploited next to identify the nature of the requirement, i.e., classify the requirement based on a pretrained classifier. This classification algorithm provides a description of the nature of the requirement which is then passed downstream to subsequent steps.
- Third, a parameter and value identification algorithm is called upon. The value identification follows from the dependency tree identification as valued parameters are identified with the "nummod" [192] modifier. For the given example, 1800 and 2 are identified as the values of interest. For each value identified, the association of a parameter is attempted. Both the parameters "distance" and "time" are identified via a mapping of the objects of prepositions associated with the numerical value to a predetermined set of parameters. The algorithm relies on the use of a set of prespecified mappings to identify standard parameters for unspecified requirement parameters. The nature of the requirement serves as an input to this algorithm to handle special cases where the parameters of the requirement are not cardinal numbers, such as in the case of take-off surface or the type of vehicle launch mechanism.

- Finally, the relationships that exist in the given description are used to identify the source node. This process of identification of the source node can enables the extraction of the validation relationship for the requirement. In the above example, the relationship extraction algorithm identifies that for requirements, distance and time, the nominal subject is the noun "*UAV*". One assumption is that there is an entity in the structures block whose name or description matches the target identified by the relationship identification algorithm. It also assumes that the identified parameters (distance, time) exist on that node in either the identified form or in one of their mappings, i.e., distance being mapped to the parameter range.

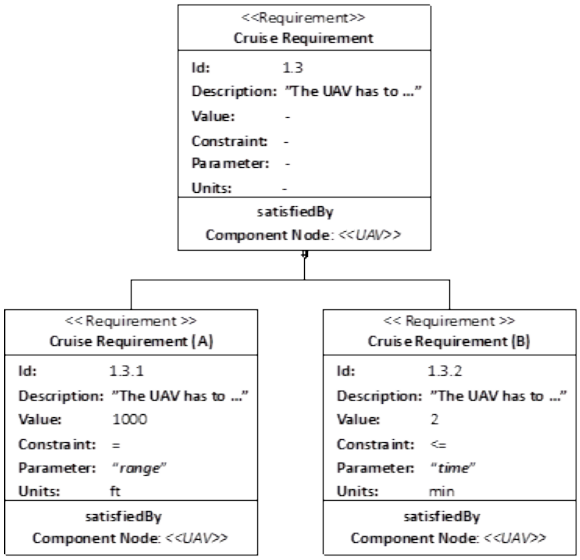


Figure 6.3: A SysML-like representation of the generated cruise requirement

With this information, the requirements tree is ready to be populated with the new subtree constructed from the specified description. The leaf nodes of this subtree are assigned a unique semantic ID and have the attributes of description, parameter, value, relationship and source. This information is then encoded into the requirements model

that is illustrated in Figure 6.3. To verify the satisfaction of the requirements, a two-stage approach is taken. First if a requirement has an associated source, parameter, relationship, and value, the evaluated value of the parameter of the source node is checked to see if the requirement is satisfied. If the requirement is satisfied, the satisfaction of all its children is evaluated. If all the child requirements are satisfied, then the evaluation would proceed to a sibling requirement node. This recursive approach is employed to identify how requirements flow from the leaves to the root of the tree, therefore providing users with a clear picture of the source of infeasibility, if any, as illustrated in Figure 6.4.

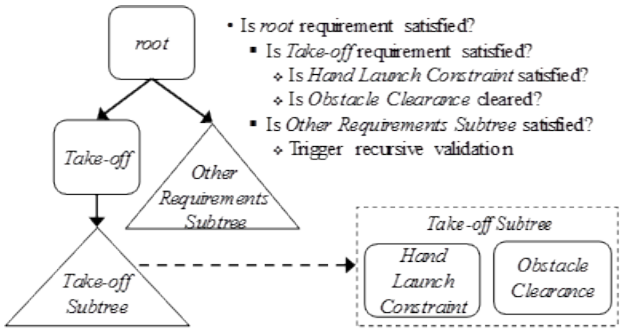


Figure 6.4: Process flow for the validation of requirements through recursive traversals of the requirements tree

An example of the set of requirements associated with the design is illustrated in Figure 6.5. The figure demonstrates the result of the decomposition of requirements from that of the definition of the mission requirements to the sub-requirements using the process described above.

Requirement 1: ***Mission Requirement*** • The UAV shall complete successfully the mission as per the mission description.	*Requirement 1.1:* ***Take-off requirement*** • The UAV shall be hand-launched and shall clear the 10 m high forest canopy within 50 m.	*Requirement 1.1.1:* ***Take-off requirement (A)*** • The UAV shall be hand-launched.
		Requirement 1.1.2: ***Take-off requirement (B)*** • The UAV shall clear an obstacle of 10 m.
		Requirement 1.1.3: ***Take-off requirement (C)*** • The UAV shall satisfy the climb requirement within 50 m.
	Requirement 1.2: ***Climb requirement*** • The UAV shall climb to a height of 100 m above ground level at a rate of 1.5 m/s within 600 m.	*Requirement 1.2.1:* ***Climb requirement (A)*** • The UAV shall climb to a height of 100 m above ground level.
		Requirement 1.2.2: ***Climb requirement (B)*** • The UAV shall climb at a rate of at least 1.5 m/s.
		Requirement 1.2.3: ***Climb requirement (C)*** • The UAV shall satisfy the climb requirement within 600 m.
	Requirement 1.3: ***Cruise requirement*** • The UAV shall fly a distance of 1,800 m within 2 min.	*Requirement 1.3.1:* ***Cruise requirement (A)*** • The UAV shall fly a distance of 1,800 m.
		Requirement 1.3.2: ***Cruise requirement (B)*** • The UAV shall satisfy the cruise requirement within 2 min.
	Requirement 1.4: ***Loiter requirement*** • The UAV shall loiter over the point of interest for at least 5 min.	
	Requirement 1.5: ***Cruise requirement*** • The UAV shall fly a distance of 2,700 m to a retrieval point within 3 min.	*Requirement 1.5.1:* ***Cruise requirement (A)*** • The UAV shall fly a distance of 2,700 m.
		Requirement 1.5.2: ***Cruise requirement (B)*** • The UAV shall satisfy the cruise requirement within 3 min.
	Requirement 1.6: ***Landing requirement*** • The UAV shall land within 15 m on a turf field and without damage.	*Requirement 1.6.1:* ***Landing requirement (A)*** • The UAV shall land within a distance of 15 m on a turf field.
		Requirement 1.6.2: ***Landing requirement (B)*** • The UAV shall not be damaged during landing.
	Requirement 1.7: ***Payload requirement*** • The UAV shall carry a payload of 0.15 kg.	
	Requirement 1.8: ***Weight requirement*** • The gross weight of the UAV shall not exceed 5kg.	
Requirement 2: ***Manufacturing Requirement*** • The UAV shall be manufactured within 48 hours.	*Requirement 2.1:* ***Manufacturing requirement*** • The manufacturing time for the UAV shall not exceed 48 hours.	

Figure 6.5: An example of the requirements decomposition that guides the design process of the UAV

6.1.2.2 Structures Modeling

The function of the structures package in the SysML paradigm is to describe the constituent elements of the system being modeled and also the relationships that exist between the elements of the system. The nature of UAV design is such that the elements of the system are typically obtained through off-the-shelf components with the components being chosen based on the design performance and the requirements defined. A design variant is defined as a combination of a set of compatible elements forming the system and a set of values for the scalable design parameters. The purpose of decomposing the vehicle is not only to enable the transition from one design variant to another by replacement of one modular component with another, but also to establish the interface specifications for these components. Thus, the structures package is formed by a

physical and functional decomposition of the UAV which results in a set of branches along which the decisions can be made. The first branch is that of the architecture of the UAV. The current research work limits the consideration of the architectures to the fixed wing and multicopter UAVs. Each one of these architectures are further decomposed into a set of subdivisions namely the agility- or endurance-focused vehicle in fixed wing group and the hexa- and quad-rotor vehicle in the multicopter group. The other branch of the *"structure tree"* is that of the components that are placed on the UAV. These components choices list the set of possible off-the-shelf alternatives that have to be loaded onto the vehicle in order to provide the necessary functionality, such as propulsion, electronics etc. A functional decomposition of the subsystems of the vehicle is carried out. The functional decomposition identifies the subsystems of the vehicles as being the propulsion subsystem, the flight-control subsystem, and the payload subsystem. These subsystems are then physically decomposed to identify the constituent components. The propulsion system is decomposed into motor, battery, electronic speed controller, and propeller. The flight-control subsystem is decomposed into servo motor, electronic speed controller, battery, and control surfaces. When multiple subsystems share components, interface relations are established to enforce the shared relation between these components. Similar exercises are undertaken for the payload subtree in order to generate the structure tree. The final decomposition of the system is carried out with the identification of the manufacturing alternatives for the vehicles. In the current analysis, it is assumed that the vehicles are additively manufactured through commercial 3D printing devices for which a set of three printers are selected. The complete decomposition of the system without the various instances of the components is illustrated in Figure 6.6.

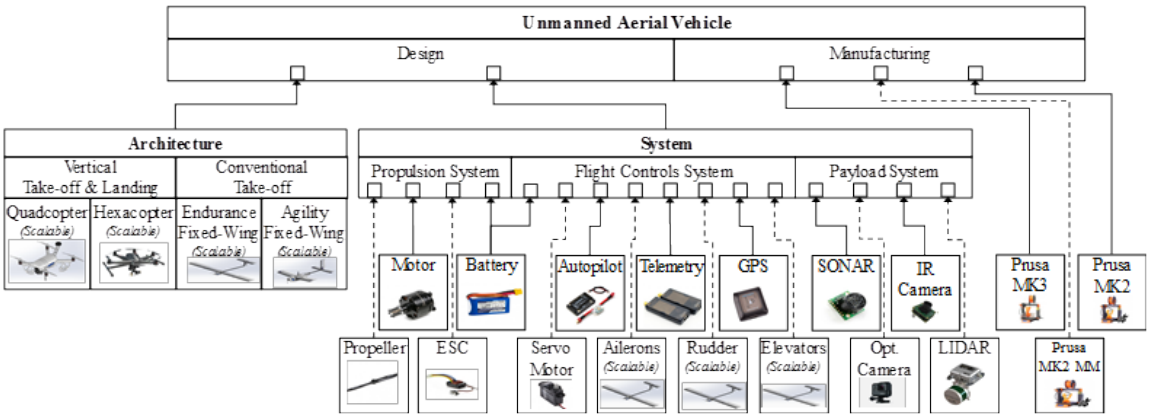


Figure 6.6: The decomposition of the UAV indicating the subsystems and the components that have to be modeled in SysML

The decomposition of the vehicle provides an indication of the architectural and design space of the vehicle, but these spaces are constrained by the compatibilities that exist between the components. For example, the multicopter UAVs do not require any servo motors on board in order to operate. This can be represented as a compatibility constraint between the two subsystems. This represents a physical incompatibility between two components. On the other hand, a physics-based compatibility relationship can also be established. For example, a certain motor may require a specific nominal voltage. An interface constraint restricting the selection of batteries unable to provide this voltage can be established for this motor, thereby reducing the set of possible variants. Having decomposed the entire system, key attributes of components are gathered and classified into two groups. The first contains commercially-off-the-shelf alternatives while the second contains specifically designed parts. Databases containing specifications are established for the commercial off-the-shelf alternatives such as batteries, motors, propellers, electronic speed controllers and payloads. These configurable databases are then used to populate the leaf nodes for the appropriate component node. The final step in the decomposition is the identification of the design parameters for the components

classified as being specifically designed. These, in addition to the design parameters at the architectural level, dictate the set of scalable parameters that are to be considered during the design process. Figure 6.7 illustrates an example of the SysML-like representation that is generated for the one of the components that constitutes the vehicle and its decomposition into a subset of its alternatives. Table 6.1 on the other hand illustrates the decomposition of each of the decomposed components that forms the basis for the extracted knowledge and that of the representation scheme that is employed by the learning algorithm.

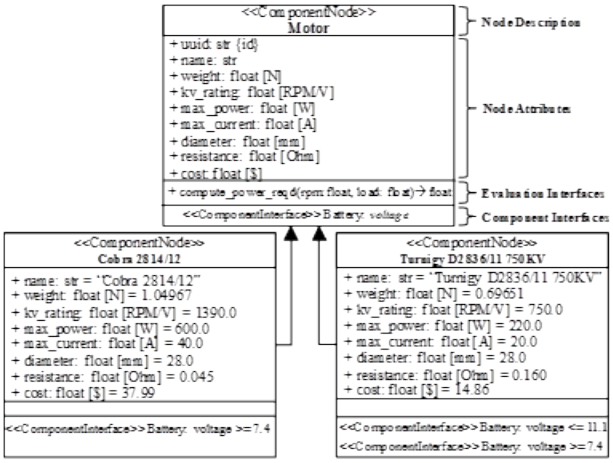


Figure 6.7: SysML-like representation of a decomposed component (motor) of the UAV with two of its instance alternatives

Table 6.1: Specification of the attributes for each component of the structure tree of the UAV system

Battery	Motor	Propeller	ESC	Servo	Payload
Brand	Brand	Brand	Brand	Brand	Weight
Capacity	KV Rating	Weight	Max. Current	Weight	Power Draw

# Cells	# Cells	Diameter	Cost	Speed
Discharge Rate	Max. Current	Pitch	# Cells	Torque
Mass	Max. Power	# Blades	Mass	Cost
Cost	Cost	Cost		
Voltage	Weight			
	Resistance			

Table 6.1 (Continued)

6.1.2.3 Processes Modeling

The new processes package takes over the function of the behavior and the parametrics packages of the traditional SysML. This new package serves the function of representing the relations that define the behavior guiding the parameters associated with the components and elements defined in the decomposed system. In order to simplify the computational representation of the design workflow, a tree data structure is retained. This is a result of the multi-layered abstraction of the design process. This multi-layered abstraction or hierarchy is divided into three levels:

- Workflows, which define the evaluation sequence of nested entities and the data flow between them
- Disciplines, which indicate the disciplinary analysis that can be performed
- Analyses, which represent the set of alternatives within any disciplinary analysis

In order to represent the processes involved during engineering design, an attempt is made to mimic an established standard for their representation using design structure

matrices. Design structure matrices provide a representation of the flow of information between disciplines and are visually appealing for complex workflows where numerous parameters passed around.

The workflow element of the framework can be viewed as a container which is associated with an algorithm. A library of algorithms is provided such that a variety of numerical computations can be performed, such as design of experiments, numerical optimization, and uncertainty quantification. This library feature is exploited to perform both the capability exploration and the uncertainty quantification. Prior to defining the characteristics of the workflow, it is nevertheless necessary to define the actual process that the algorithm will operate on. The workflow can host a set of interconnected disciplines, a set of nested workflows, or a combination of both. The discipline functions as a container for a set of analysis owing to its multi-fidelity nature. For a discipline to be evaluated, there has to be exactly one analysis that is active at any instant of time. If a discipline does not need to be evaluated, logic can be included to deselect any constituent analysis at which point the execution of the discipline is skipped until a selection is turned back on again. The final layer of the hierarchy is the analysis node. The analysis represents the actual computation that is performed in the workflow. These, in addition to the workflows, represent the executable components of the framework.

The executable nature is achieved by the registration of an evaluation source for every evaluation node which is represented by the created workflows and analyses. In the case of the workflow, the evaluation source would be an algorithm, while in the case of the analysis the evaluation source would represent some numerical computation. These evaluation nodes are then linked to the evaluation ports, which are abstract methods

definitions on the structure. This linkage definition ensures that attributes of the structural node are visible to the evaluation source. Optional interface specifications, in the form of value bindings, can be specified for the evaluation sources. The bindings efficiently map parameters from the evaluation to and from the registered structure node as an evaluation is undertaken. These values bindings can also map the nodes themselves in cases where multiple parameters have to be accessed, thus reducing the amount of coding necessary. Interfaces across disciplines can be defined at the workflow level such that mapping between parameter values are efficiently handled upon completion of the execution of a discipline. These parameter mappings can be defined through equations that dictate the *'interface'* between scalable parameters in the case of the UAS application.

This means of representation for the evaluable workflow enable the virtual verification of requirements with the introduction of the ability to link parameters of the analyses to those defined in the structure of the system.

6.1.2.4 Vehicle Sizing and Synthesis Workflows

The performance analysis of the system is carried out through the created workflows. In particular two separate workflows are defined, one that represents the design process involved in the design and manufacturing of a fixed-wing vehicle and the other that represents the processes involved in the design and manufacturing of the multicopter vehicle. The objective of the sizing and synthesis is the identification of the appropriate choice of the wing loading and the optimal wing and the empennage on the fixed wing vehicle and the identification of the minimum arm length necessary on the multicopter vehicle. The wing loading determines the wing area and hence in turn determines the system weight that is utilized as an indicator of the goodness of a design. In order to

estimate the wing loading associated with the vehicle, a version of the Mattingly constraint analysis [193] adapted to low-Reylond's number flight is performed. Equations are developed for the relationships between the manufacturing time, components weights and wing loading in order to generate a constraint diagram from which a suitable design can be selected. Similarly, for the case of multicopter sizing, a power-based and energy-based formulation is utilized. The process involved in the sizing and synthesis of the vehicle is explained in detail in the published work [180].

6.1.2.5 Relationship between the Sizing and Synthesis and the Artificial Agent

In both cases of the fixed-wing and multicopter design, the sizing and synthesis functions as the black-box environment that the artificial agent interacts with selecting both the components that have to be loaded onto the vehicle and the parameters associated with the size of the vehicle. From the perspective of the artificial agent, the sizing and synthesis routine serves the purpose of indicating the goodness of a selected set of decisions. In the current analysis, this goodness is measured as a function of the total weight of the resultant system, the ability to meet the defined requirements and the resultant manufacturing time of the vehicle. Thus, the artificial agent views the entire MBSE application as a black to which a set of actions are suggested and an estimate of the changes resulting from the action is generated and the reward associated with the actions undertaken are estimated as a function of the physics-based relationships represented by the sizing and synthesis module.

6.2 Extraction and Encoding of Knowledge from the MBSE application

In the case of the UAV design the purpose of the models developed in the MBSE application is the representation of the instantaneous design characteristics of the UAV.

As the process of UAV design involves the selection of a compatible set of off-the-shelf components and the selection of the scalable parameters associated with the selected architecture the knowledge extracted from the MBSE application would represent the instantaneous state of the design in terms of these variables. As the application developed utilizes Qt as its backend, the computational implementation of the knowledge extraction is greatly simplified. Qt provides the capability to listen to changes in object states through the use of signals and slots. This capability is utilized in the current research work as the model based representation of the components translates to an object-oriented implementation in the framework. Thus, as and when the state of the object is altered, for example, in the case of the system: a change in its composition, a signal can be emitted that is processed by a slot which can trigger the extraction of information from the MBSE application.

6.2.1 Extraction of knowledge

The extraction process is illustrated in Figure 6.8 which highlights the sequence of interactions that occur between the MBSE application and the external framework. In order to simplify the interactions that need to occur between the elements of the framework, the MBSE application is launched as a separate dialog of a container application. The container application is configured to interact with the database that stores the data that is to be utilized for the learning. As both applications are developed on a common platform, the exchange of data between them is simplified.

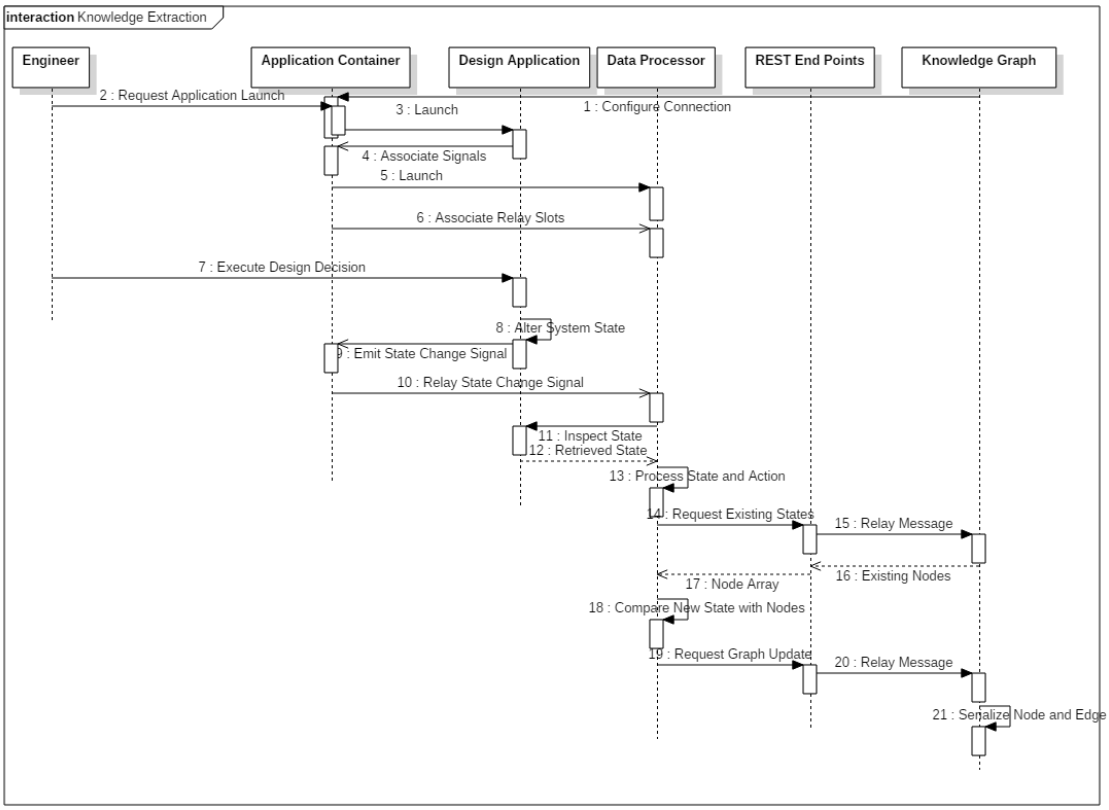


Figure 6.8: Sequence diagram indicating the interactions that occur between the elements of the knowledge extraction routine in the MBSE application

As illustrated by the figure, the slots are established on the container application to tail the system model. When any change occurs on the system model, a serialization process on the container is triggered where the contents of the system model are vectorized in order to generate a node in the database. In contrast to the current use-case, there is significantly more detail that is captured in the Siemens NX use-case in the serialization process. As the application is assumed (and designed) to be a grey-box, it is possible to access certain portions of the application through an API. This enables the extraction of information related to the models that have been specified on the system being designed. The attributes listed in Table 6.1 represent the data that is extracted by

the knowledge extraction process, with an example of the graphical representation of the extracted information shown in the Figure 6.9.

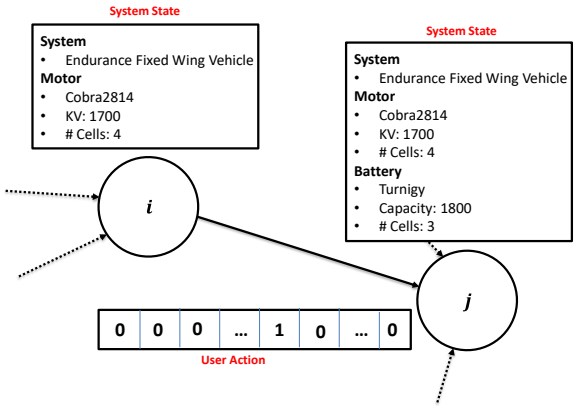


Figure 6.9: An illustrative example of the graphical representation generated of the extracted information

As observed in figure, the nodes of the graphical model represent the states of the design at different instants of time. In this case, as the states of the design represent a sequence of choices associated with the components and the scalable parameters, the nodes encode the information related to the following,

- The type of UAV architecture utilized
- The attributes associated with the components on-board the UAV architecture
- The scalable parameters for the UAV architecture

Although not shown in the graphical model, each node in the graph is associated with a list of encoded requirements, which have been encoded using the previously introduced embedding mechanism. The edges between the different states represent the action that the *"engineer"* implements. In the current use-case these are represented by either the discrete choice of an architecture of the UAV or component that is to be loaded onto the

architecture or the choice of a scalable parameter. As the design is virtual in nature, there is no restriction on the order in which the components have to be chosen, although the framework is engineering in a way such that the choice of the scalable parameter is triggered as the final decision of the artificial agent.

6.2.2 Knowledge Encoding

The encoding process utilized converts each node and edge on the generated graphical model into a vector of real numbers. As the combination of a pair of nodes and an edge represent the transition from one state to another, i.e., the decision made by an engineer, this sequence of node-edge combination forms the ideal data set to trigger the learning process. Thus a vectorized representation of the system is generated by stacking the attributes of the components on board with the indicator of the architecture and that of the scalable parameter. The action, on the other hand, in case of the selection of an attribute is represented as a discrete integer indicating the index of the component in a database of predefined set of components and in the case of the scalable parameters represents the actual value of the scalable parameter normalized between predefined bounds. Figure 6.10 represents the encoded state of an example node that is generated through the encoding mechanism utilized. The framework is implemented in a manner such that the encoding mechanism introduced is triggered by the container application and the encoded information is stored with the node on the graph database.

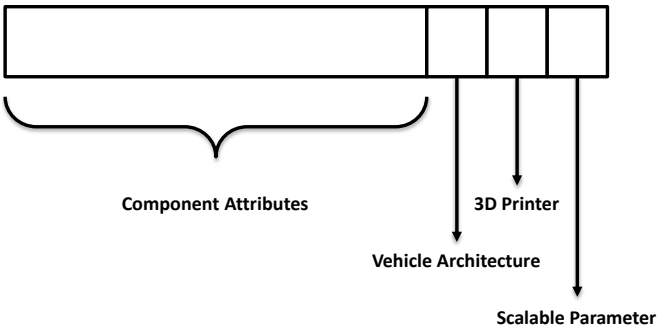


Figure 6.10: Information encoded in each node of the graph that represents the instantaneous state of the UAV design

Figure 6.11 represents a generated knowledge graph containing multiple different design variants for a single set of requirements. It is important to note that the length of each branch on knowledge graph is dependent on choice of the components as an analysis of the feasibility of the component combination is carried out as and when there is a change to the system composition. Thus, if the engineer selects two components that are infeasible as the first two decisions, then the resultant branch would only contain two nodes. Each pink node in the image represents one combination of the design, either complete or incomplete, compatible or incompatible and the edge between the nodes represents a human action in which some change has occurred to the state of the design. A sequence of node and edges when put together represents one exercise of the design process in which a certain combination has been evaluated.

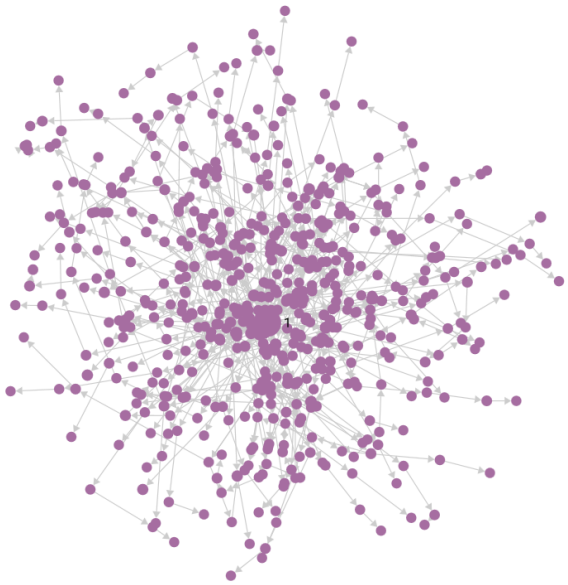


Figure 6.11: A representation of the knowledge graph representing the several different design variants that are generated through different combinations of architectures, components and scalable parameters.

6.3 Formulation of the Learning Problem

It is important to note that each design variant that is created in Figure 6.11would be associated with a set of performance metrics. The performance metrics not only indicate the capability of the design variant to meet the specified set of requirements but also the other key performance indicators that are associated with the design variant such as the total weight of the UAV and manufacturing time associated with the design. Thus, methods such a Monte-Carlo tree search [194] can be applied to identify the best combination of architecture and components, following which an optimization process can be triggered to optimize the choice of the scalable parameters. While such methods do produce optimized results, they fail to adapt to the knowledge that is gained through the process of exploration. Thus, the framework of reinforcement learning is utilized where knowledge gained in the process of exploration is used to bootstrap the estimates of performances to guide future explorations to better regions of the space. In order to

formulate the exploration problem in terms of the mathematics required for reinforcement learning it is necessary to define the contents of the state, action and the rewards associated with the exploration problem. The following passages indicate the formulation chosen for the problem of UAV design, following which the architecture of the agent utilized to perform the exploration is introduced.

6.3.1 Representation of the System State

As discussed in CHAPTER 3, the purpose of a design application is to provide a representation of the product being designed at any instant of time. In the case of the MBSE application introduced in this chapter, the system being designed is represented in a graphical format on a central database where each node in this knowledge graph represents one unique state of the system being designed. Thus, the vectorized representation of the nodes can be utilized as the representation of the state of the environment. In the current analysis, though, in addition to the specification of the design, each design is generated in response to a particular set of requirements. Thus, the requirements too, need to be encoded in the state of the system, in order to generate a complete representation of the design problem. Figure 6.12 illustrates the modified encoding of the state of the node to account for the requirements that are defined in the system. In the implementation of the framework, the state of the system is ordered in manner given by the specification indicated in Table 6.2 which results in a $(2037, 1)$ dimensional vector to generate the representation of the state of the system.

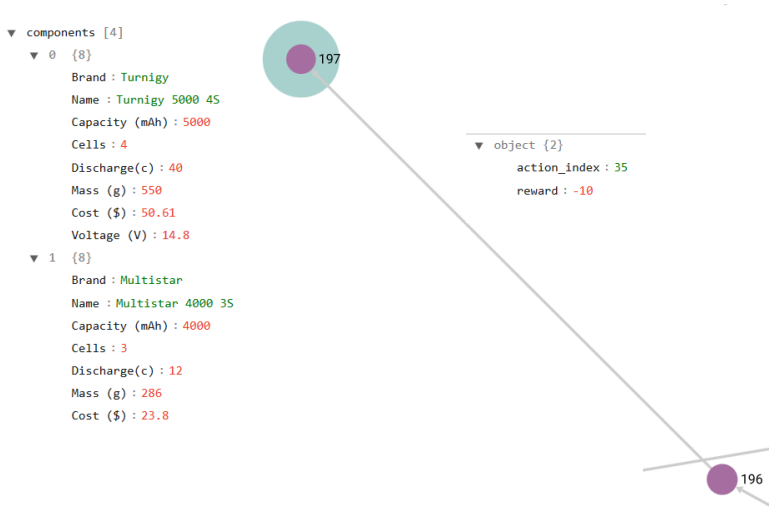


Figure 6.12: Representation of the state of the system comprised of the attributes associated with the components on board the system, the index associated with the active architecture, the scalable parameter associated with the architecture and the requirements defined for the design problem

Table 6.2: Composition of the ordered state vector

Vector Source	Interpretation	Data Type	Dimension
Motor	Attributes of the motor that is on board the UAV	Real	(7, 1)
Battery	Attributes of the battery that is on board the UAV	Real	(7, 1)
Propeller	Attributes of the propeller driven by the motor	Real	(6, 1)
ESC	Attributes associated with the electronic speed controller	Real	(5, 1)

Servo	Attributes of the servo motor onboard the UAV	Real	$(4, 1)$
Payload	Vector representing the presence of a certain type of payload	Integer $\epsilon\ [0, 1]$	$(5, 1)$
Architecture	Index of the active architecture	Integer $\epsilon\ [-1, 3]$	$(1, 1)$
Machine Id	Index of the 3D printer used for manufacturing	Integer $\epsilon\ [-1, 2]$	$(1, 1)$
Scalable Parameter	Represents the value of the scalable parameter associated with the architecture	Real	$(1, 1)$
Requirements	Embedding of the requirements set	Real	$(20, 100)$

Table 6.2 (Continued)

The formulation of the requirements utilizes a 20 dimensional embedding of the requirements description an allocation of 100 requirements for the design problem. In the scenario that are fewer than 100 requirements for the complete design problem, the elements of the requirements matrix that is not populated with the requirement embedding are assigned a value of -1. The requirement matrix is then vectorized to generate a 2000 dimensional array that is stacked onto the state vector to generate the modified state vector illustrated in Figure 6.12.

6.3.2 Representation of the Actions

The actions are encoded in two ways, one for the discrete choices that are contained in the environment (such as the choice of components and architecture) and the other for the choice of the continuous parameters, such as the scalable parameters. The encoding of the actions of the discrete parameters is represented as a one-hot vector in which each vector entity represents on choice of a component or architecture. In the analysis performed, there are two different scenarios considered. The first scenario involve 44 different alternative and the second, which is an analysis of the scalability of the approach, involves 152 alternatives, divided amongst the components as indicated in Table 6.3. Thus, the action space for the two scenarios is represented by $(44, 1)$ and $(152, 1)$ dimensional vectors respectively.

Table 6.3: Subdivision of the discrete alternatives in the two scenarios analyzed

Scenario Id	Battery	Motor	Propeller	ESC	Servo	Payload	Architecture	Machines
1	10	11	6	5	5	5	4	3
2	50	25	15	25	25	5	4	3

On the other hand for the continuous scalable parameter, a single real number is utilized to represent the value of the parameter within predefined bounds. The bounds are as indicated in Table 6.4.

Table 6.4: Parameter bounds for the scalable parameters

Scalable Parameter Type	Parameter Bounds
Fixed Wing Vehicle: Wing Area	$0.05 - 0.70\ m^2$
Multicopter Vehicle: Arm Length	$5\ -\ 12\ inches$

6.3.3 Reward Formulation

The primary indicator of a goodness of a decision in the reinforcement learning field is the reward metric. In the current analysis, a complex formulation of the reward is chosen such that the goal of the trained artificial agent would be to choose a design that meets all the requirements posed by the engineer and to minimize the total weight and manufacturing times associated with the design. The availability of the virtual verification in the implemented MBSE application provides an indication of the former and the ability of the MBSE framework to associate metrics from the evaluation processes to the structural elements provides the necessary key performance indicators such as the system weight and 3D printing times. For intermediate actions that do not result in a complete evaluable design, an evaluation of the compatibility of the system is undertaken which indicates if the choices of the components on board the system are compatible and in the scenario that they are not compatible, the episode is terminated with a large penalty. Each compatible selection that results in an incomplete design is associated with a reward of -1.0. As the goal of the agent is to maximize the total observed reward, this ensures that the agent does not repeatedly select the same component. The overall formulation of the reward metrics is as given below.

$$Reward = f(State, Action)$$

$$= \begin{cases} -10, & State\ is\ not\ compatible \\ -1, & State\ is\ compatible, but\ design\ is\ incomplete \\ -5, & State\ is\ compatible\ and\ complete, \\ & but\ requirements\ are\ not\ satisfied \\ g(w_{tot}, t_{manf}) > 0, & State\ is\ compatible\ and\ complete\ and \\ & the\ requirements\ are\ satistfied \end{cases}$$

6.3.4 Learning Algorithm and the Agent Architecture

With the above formulation of the reinforcement learning problem, there are two possible alternatives for the implementation of the learning algorithm. The first is the utilization of a hierarchical reinforcement learning framework [195], [196] that distinguishes between the discrete and continuous parameters with the continuous choices being made on a higher hierarchy than the discrete choice or the utilization of a sequential decision model, i.e., an agent which is composed of two distinct agents each optimized for their own tasks, i.e., the first agent is optimized for the task of architectural design while the second is optimized for the task of scaling an architecture. This formulation is similar to the options framework [197] introduced in the hierarchical reinforcement learning where a semi-Markov decision process is constructed from an existing Markov decision process. The second approach is favoured over the first for two primary reasons,

- The agents are independent of each other implying that the training of each agent can occur independently. This is in contrast to the hierarchical framework where internal agents view a subset of examples than the external agent making the training of internal agents quite difficult.

- From an implementation perspective, this forms a simple workflow as illustrated in Figure 6.13: Process flow involved within an agent in the creation of a design variant, where the continuous agent is only triggered when the discrete agent has made a complete selection of the alternatives. As the goal of the research work is the application of existing reinforcement learning techniques for the purpose of design automation, the simpler implantation offered by the sequential model is chosen over the hierarchical reinforcement learning framework.

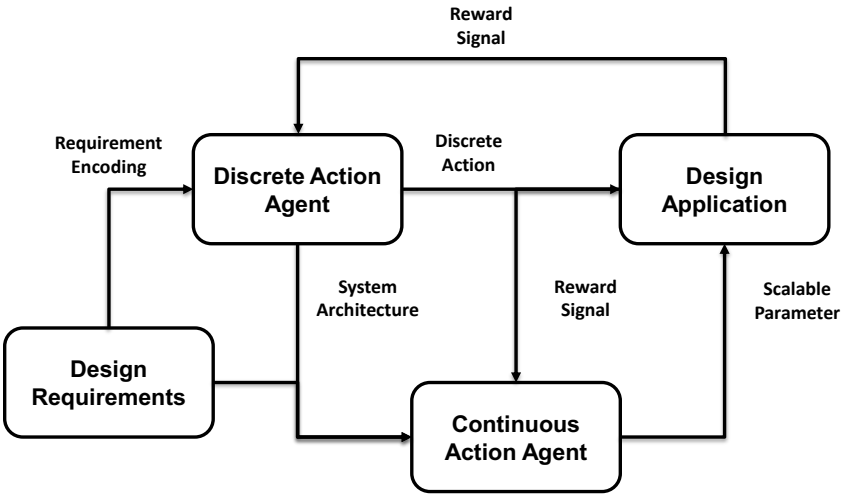


Figure 6.13: Process flow involved within an agent in the creation of a design variant

With this formulation, the rewards observed by the two agents are modified as follows,

$$Reward_{continuous} = f(State, Scalable\ Parameter) = g(w_{tot}, t_{manf})$$

$$Reward_{discrete} = f(State, Architecture)$$

$$= \begin{cases} -10, & State\ is\ not\ compatible \\ -1, & State\ is\ compatible, but\ design\ is\ incomplete \\ -5, & State\ is\ compatible\ and\ complete, \\ & but\ requirements\ are\ not\ satisfied \\ 5 + g(w_{tot}, t_{manf}) > 0 & State\ is\ compatible\ and\ complete\ and \\ & the\ requirements\ are\ satistfied \end{cases}$$

6.3.4.1 Agent architecture and the learning algorithm

The other consideration that is to be made, in addition to the architecture of the learning process, is the architecture of the agent. In this case, the agent is comprised of two independent models, the first that recommends the discrete choices and the second that recommends continuous scalable parameters. In both cases, the input state vector of the neural network is represented by a $(2036, 1)$ vector which is a subset of the $(2037, 1)$ element state vector introduced in the previous section. The element corresponding to the scalable parameter is dropped from consideration as it is now a part of the action space of the second network.

In order to train the discrete action network, the double deep Q-learning (DQN) [198] algorithm without duelling is utilized. The action space for this network is the discrete action space represented by the $(44, 1)$ choices in the first scenario or the $(152, 1)$ choices of the second scenario. Thus the network condenses an input space of $(2036, 1)$ to generate an output of $(n_{actions}, 1)$. To enable sufficient learning, a 5 layer deep network is considered and the architecture of the network is illustrated in Figure 6.14. The intermediate layers are activated by the rectified linear units [199] with the first two layers containing 1000 neurons with the third layer contains 500 neurons and the final two layers contain 250 and 125 neurons respectively. The final layer is activated

using linear units in order to account for the rewards being both negative and positive and also greater than 1. As required by the formulation of the Double DQN algorithm both the target and the prediction networks share the same network architecture. The number of parameters associated with this networks is that are trained during the learning process is 3,700,699 in the first scenario and 3,714,277 in the second. The model is trained using the Adam algorithm [200] with a learning rate of 0.001 annealed to 0.0001 at the end of the learning process, i.e., 1,000,000 or 10,000,000 steps as per the scenario.

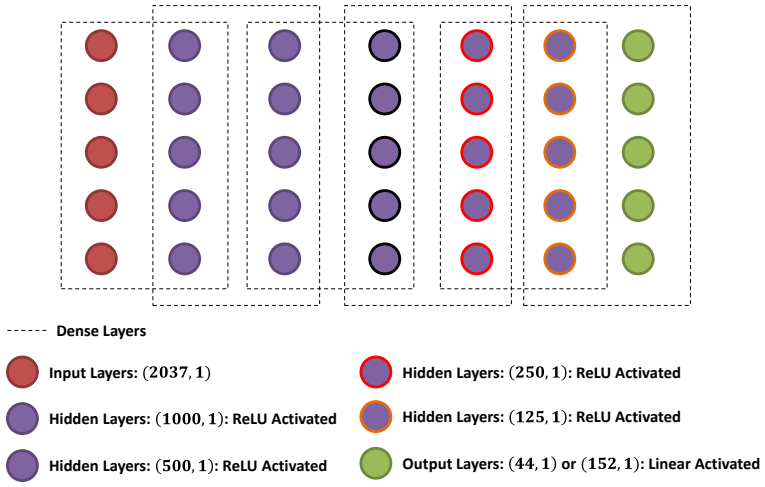


Figure 6.14: Neural network architecture of the Double DQN agent used to predict or recommend discrete architectural parameters

On the other hand, for the network handling the prediction of the continuous parameters the deep deterministic policy gradients [201] algorithm is used to train the model. Both the actor and the critic network required by the algorithm share the same architecture, where the input state vector still retains the $(2036, 1)$ dimension. The networks utilize three hidden layers that are activated once more with rectified linear units having 1000, 500, and 100 neurons respectively. These layers are connected using a dense connection in between them and with a dense connection to the action unit that is

represented by a single neuron. The output neuron utilizes a linear activation. The architecture of the agent with both the actor and critic networks is illustrated in Figure 6.15. The continuous network too is trained using the Adam algorithm with a similar setting of the optimization hyperparameters with the optimizer operating on a 3,695,251 weight parameters. In the case of the continuous parameter, the input state vector would represent a complete design which has all the necessary components defined, architecture selected and the requirements appended to the encoded vector.

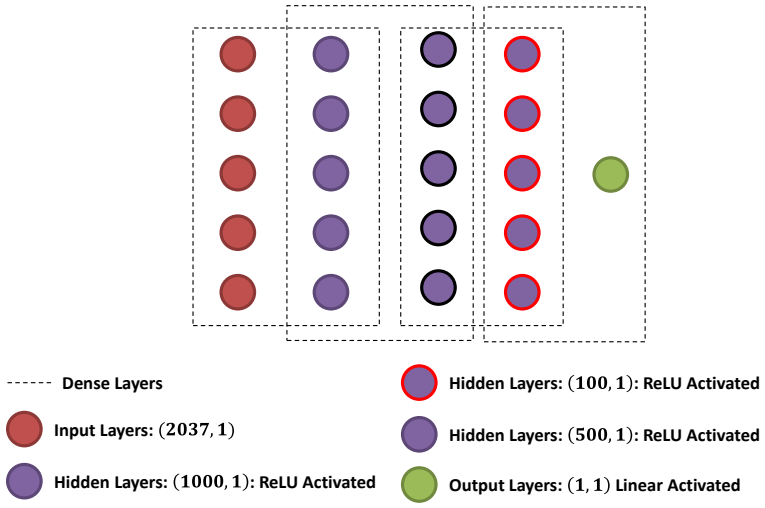


Figure 6.15: Agent architecture for the DDPG algorithm by both the actor and the critic networks for the prediction or recommendation of scalable parameters

6.4 Results of the Application of Machine Learning

The previous sections have introduced the problem formulation associated with the reinforcement learning setup, the network configuration associated with the learning setup and the configuration of the hyperparameters related to the learning process. With these configurations, a set of five analyses are carried out to evaluate the performance of the reinforcement learning algorithm for the task of automation of engineering decisions in the MBSE design setting.

6.4.1 Analysis 1: Utilization of reinforcement learning without demonstrations

The first analysis compares the performance of a genetic algorithm (GA) [202] for the purpose of architecture selection to that of the reinforcement learning agent that self-learns without any demonstrations on the first scenario of the UAV design. The formulation of the reward is modified such that the target of the learning process is to minimize the total weight of the resultant system that meets all the specified requirements. The choice of the scalable parameter is made using same heuristics, i.e., smallest possible wing area and smallest possible arm length, in both the algorithm so as to ensure appropriate comparisons. In the case of the GA, a population size of 300 is utilized and a chromosome size of 10 is utilized. In this case, the chromosome of 10 represents the sequence of choices performed by the engineer, with the genetic algorithm being simulated for at least 1,000,000 steps. The objective function for the GA is coded to be the same as that of the Double DQN algorithm, i.e., maximization of the cumulative reward from each step. In terms of the evaluation of the cumulative reward for the GA, when the algorithm generates a chromosome which results in a feasible and complete design or an infeasible or an incompatible design prior to the 10 parameters, the analysis is terminated with an indication of the reward as per the formulation of the reinforcement learning problem.

The results of the analysis are averaged over 25 samples and the results of the first 1,000,000 steps are presented in Figure 6.16 and Table 6.5: Performance comparisons of the Double DQN algorithm and the GA. It is observed that without human demonstrations, the genetic algorithm finds feasible designs faster than the reinforcement learning agent, but in most cases, the reinforcement learning agent successfully find

either the same or better designs that the genetic algorithm. This can be attributed to the fact that the reinforcement learning agent utilizes an epsilon-greedy exploration strategy that theoretically guarantees improvement of results with increase in the number of samples.

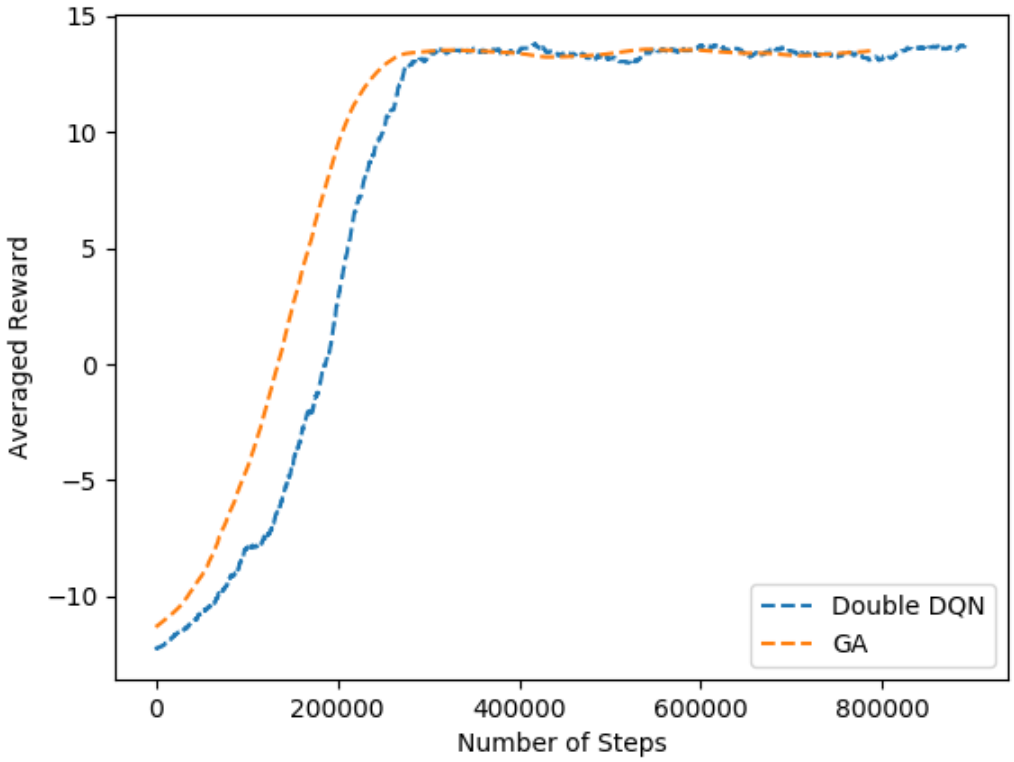


Figure 6.16: Averaged performance summary of the Double DQN algorithm in comparison to a GA

Table 6.5: Performance comparisons of the Double DQN algorithm and the GA

Algorithm	Best Reward	Mean Reward	Worst Reward	Standard Deviation	# Steps to Best
GA	**19.9661**	**14.1382**	**11.1947**	**2.2543**	**5103**

Double DQN	**21.478**	**15.99**	**9.056**	**4.0252**	**5194**

Table 6.5 (Continued)

6.4.2 Analysis 2: Replication of human level performance through expert demonstrations

The second analysis deals with an attempt to replicate *human*-level performance through the use of pre-training on the Double DQN algorithm. With the introduction of the pre-training, the Double DQN algrorithm is replaced with the Deep Q-Learning from Demonstrations (DQfD) [155] algorithm, but the architecture of the network, the configuration of the optimizer and the formulation of the problem are retained. Due to its ability to generate large quantities of data, a GA is utilized to generate the data for the learning process and to simulate the presence of two distinct engineers. The generated data is sorted by the reward metric and the database is split into two halves. It is assumed that the first half of the data is generated by an expert engineer where the generated rewards are high, and that the second half of the data is generated by a novice engineer with low reward values and inefficient selections. As with the previous case, the objective of the GA is to maximize to cumulative reward observed by a certain member of the population. The DQfD algorithm inherently relies on the utilization of the concept of Prioritized Experience Replay [154], a concept that is modified to account for the incorporation of data from multiple different engineers or sources. The framework utilizes a weighting factor associated with each type of engineer to alter the probability of sampling a demonstration from that engineer such that samples demonstrated by the expert engineer is preferred over to that demonstrated by the novice engineer. The current analysis implements the modification of the prioritized experience replay sampling

routine to account for the presence of demonstrations from multiple sources, with one being preferred over the others. This implementation utilizes a value of 0.75 for the value of h_1 (expert engineers) and a value of 0.25 for that of h_2 (novice engineers) so as to guide the sampling towards the data generated by the expert engineers.

In order to extend the use case to the problem of self-exploration the database is divided into an exploratory database and a demonstration database, with the sampling during the pre-training phase occurring only from the demonstration database. In the exploratory learning phase the sampling gradually anneals from the demonstrated database to the exploratory database.

The results of the analysis of replication of human-performance through pre-training are demonstrated in Figure 6.17 where the impact of the number of pre-training steps and the amount of data are investigated on the level of performance. Each analysis is repeated 25 times in order to account for stochasticity in the generate data and the learning algorithm. It is observed as the amount of data increases, there is a necessity for additional training in order to replicate the demonstrated *"human-level"* performance. The results indicate the algorithm is able to replicate demonstrated performance in most of the simulated cases.

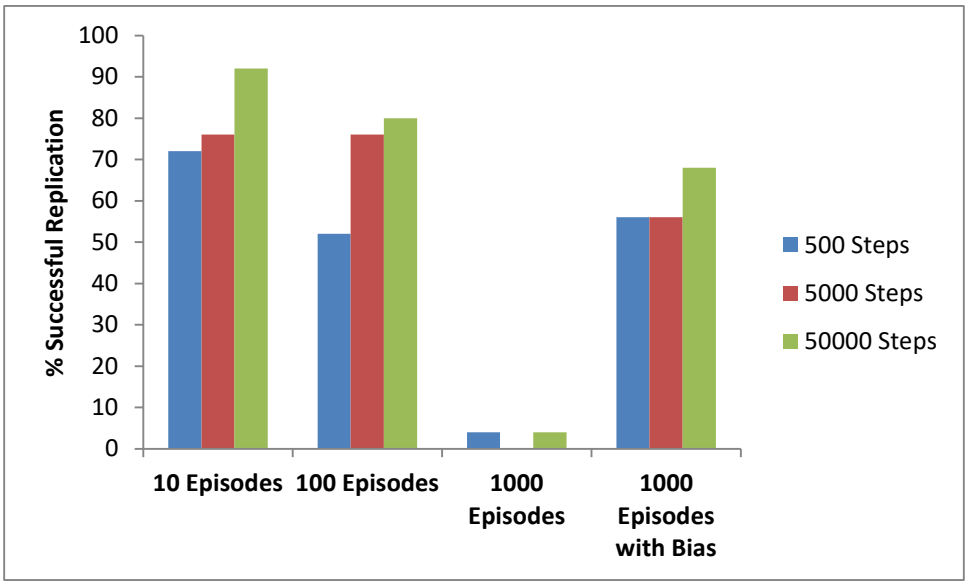


Figure 6.17: A comparison of the number of episodes and the amount of data to the impact of successful replication of human-behavior

It is important to note that in the case of the 1000 episodes, the algorithm is unable to replicate human behaviour as the quality of data contained in the demonstrations are quite bad. Out of the 1000 demonstrated episodes, only 42 episodes on average were successful in finding a compatible and feasible design. The final set of samples account for the newly introduced biased training routine in which the samples generated by group 1are sampled more frequently than that of the group 2.

6.4.3 Analysis 3: Improvements over human level performance

The third set of analysis attempts to demonstrate that the reinforcement learning framework once pre-trained can generate design variants that are better than the once observed in the pre-training batch. This analysis builds on the results observed in the second analysis with the model being pre-trained with varying number of demonstrations for a given number of steps. Having pre-trained the model, the analysis launches a self-learning period in which the model attempts to improve the observed designs by finding

better alternatives. The analysis first demonstrates the capability to find better designs than that demonstrated in the training phase as illustrated in Figure 6.18, where the results of 10 samples are averaged, with the best observed result shown in the figure to illustrate the capability for improvements over the demonstrated data. The impact of number of demonstrated pre-training steps is also shown in Figure 6.18, with the agent having larger number of pre-training steps demonstrating more rapid progression in the improvement over that of the one with fewer pre-training steps. The number of training steps beyond that of the pre-training is set to 20,000 and the modified DQfD algorithm with sampling from experienced and demonstrated buffers is implemented.

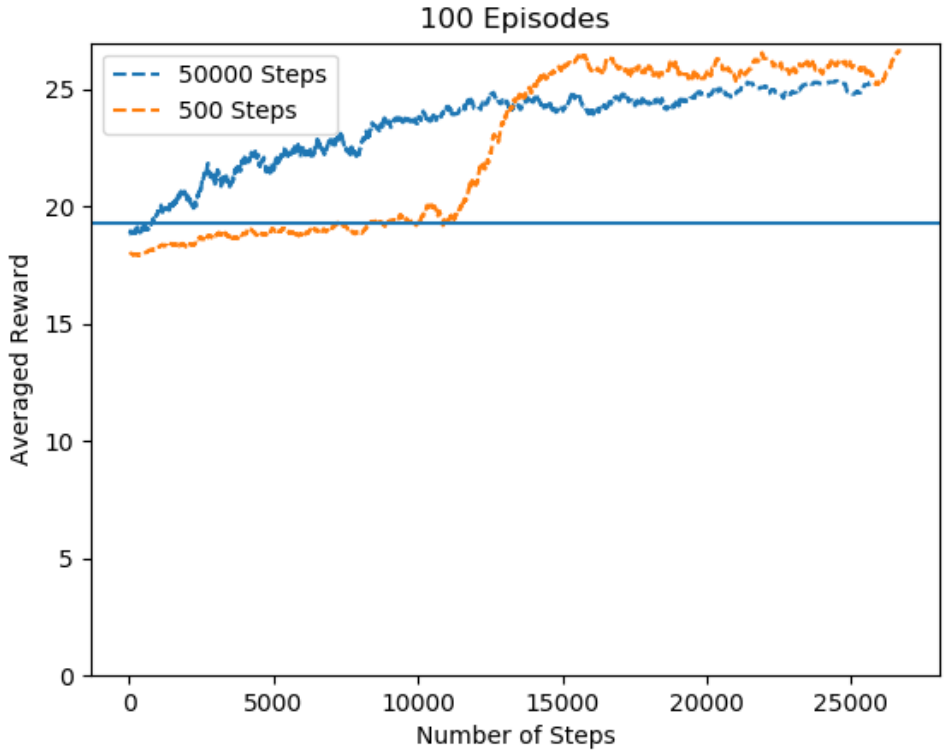


Figure 6.18: Demonstration of the capability of the algorithm to identify designs that outperform the best demonstrated design with a comparison of the learning rates associated with the number of pre-training steps utilized.

As evident from Figure 6.18, the algorithm demonstrates the capability to find designs that are better than the demonstration rather quickly when the best demonstrated data is of low quality, but it takes considerable effort to explore better demonstrations when the quality of demonstrations are high.

6.4.4 Analysis 4: Transfer of knowledge to modified requirements

The final analysis addresses the capability of the agent to adapt to scenarios that it has not observed in the past. This is carried out in two means, first one in which the requirements are modified during the training phase and the second in which the structure of the system is modified. The analysis alters only a select few requirements in order to evaluate the capability of the framework to generalize across varying requirements. The process established to carry out the analysis first involves training the agent to a varying set of requirements so as to enable the neural network to develop a model across the requirements space. The new set of requirements is then exposed to the agent to evaluate its capability to predict a feasible design without any additional training.

In order to simulate the changes in requirements, a set of 100 requirements are generated by altering the baseline set of requirements illustrated in Figure 6.5 by altering both the sets of cruise requirements in terms of the distance and the duration, the climb requirement in terms of the rate-of-climb and finally the manufacturing requirement in terms of the allowable time to manufacture. A Latin hypercube sampling [203] is utilized to generate the modified requirements associated with these parameters with the limits specified in Table 6.6.

Table 6.6: Limits associated with the requirements for Latin Hypercube sampling

Parameter	Bounds
Cruise Requirement 1: Distance	1300-2000 m
Cruise Requirement 1: Time	90-150 sec
Cruise Requirement 2: Distance	2000 – 3400 m
Cruise Requirement 2: Time	150 – 210 sec
Climb Requirement: Rate-of-climb	1.2 – 1.8 m/s
Manufacturing: Time Requirement	48 – 96 hrs

Having generated the new requirements, the embeddings associated with the requirements are altered to reflect the new set of requirements and the process of exploration is triggered. The database is then updated with the new designs and the training is triggered. In order to evaluate the performance of the design, a set of 25 new requirements from within the requirements hypercube are generated and the performance of a pre-trained agent is evaluated. The goal of the algorithm is to automatically adapt to the new set of requirements without the necessity to retrain the agent. The performance of the agent is illustrated in Table 6.7. As evident from the table, the agent is able to adapt to 40% of the new requirements cases. To further evaluate the capability of the agent, the agent is permitted to self-learn on the 25 new requirements for 20,000 steps. In this case, as demonstrated in the same figure, a majority (52%) of the 25 requirements yield a valid

design within the permitted number of steps, thereby demonstrating the capability of the algorithm to adapt to new requirements within a certain requirement space.

Table 6.7: Results observed from the analysis of 25 samples generated from altered set of requirements

Scenario	Percentage of feasible designs identified
Without additional training	40
With 20,000 additional training steps	52

6.5 Discussion

6.5.1 Achievements of the current work

- The developed framework demonstrates the capability to apply reinforcement learning to the problem of automation of engineering decisions in an engineering application. The problem of UAV design is chosen due the simplicity in the models involved and the agents developed demonstrate the capability to perform at a level similar to that of a well-established optimization algorithm.
- The developed framework demonstrates the capability to adapt to demonstrated data, such that in the case of a real engineering problem where significant amount of data exists for a certain problem, the algorithms and the processes developed would be readily applicable. The represents the capability to replicate human-level decision making in a simplified environment.
- The developed framework demonstrates the capability to exceed the performance of the demonstration through self-learning thereby established the sufficient condition

for the incorporation of artificial intelligence in the engineering decision making. An rigorous analysis of the sensitivity of the learning process is performed to evaluate the impact of the amount of pre-training steps and the amount of data available during the pre-training stage.

- Finally, the framework evaluates the capability of the algorithm to adapt to cases that it has not been demonstrated through the requirements modification use-case. It is observed that the agent is able to transfer knowledge from the trained set of requirements to a previously unseen requirement. This demonstrates that the agent is robust to changing requirements where in there is little retraining necessary in order for the agent to adapt to the new scenario.

6.5.2 Limitations of the developed framework

- The use-case chosen is of the nature where the number of decisions that is to be made are quite small in nature. This may contribute significantly to the success of the algorithm, but it is argued that as the use-case is representative of a realistic design problem, the results observed here would be transferrable to other design problems as well. There would be necessity to evaluate the performance of the methodology and the agents on cases that involve several thousands of design parameters, such as the 3D design problems.
- The use-case assumes an ability to extract, represent and encode the knowledge contained in a design system, a task that is not simple. The second use-case considers this problem in more detail where a more complex design application, such as a CAD system represented by Siemens NX is considered.

- The use-case fails to demonstrate the transferability of the results generated by the analysis. In particular, there is no evaluation performed on the capability to transfer knowledge gained by an agent in one problem to another. The current framework assumes that a new agent would have to be trained for the new problem, which may be quite expensive depending on the nature of the problem.
- The use-case also fails to evaluate alternate neural network architectures or reinforcement learning algorithms that can be utilized to train the learning agent. Additionally, alternate reinforcement learning frameworks can be leveraged to improve the training performances demonstrated by the current use-case.
- Finally, there is mathematical analysis of the results that are generated. This is an open question in the field of reinforcement learning and is not tackled by the current research work. The analysis of the performance of the framework and the agents developed are limited to being empirical in nature.

CHAPTER 7. USE-CASE II: SIEMENS NX

The current chapter introduces the second use case in which the developed framework is applied for the automation of engineering decisions in a complex black-box system. The purpose of the use case is to demonstrate the capability of the framework to capture engineering decisions in complex environments and replay these decisions in a recommendation mode of operation to imitate the behavior of a design engineer. The use case establishes an approach for the extraction of knowledge from a complex computer-aided design (CAD) system and an approach for the encoding of the captured knowledge in a manner in which it is usable by machine learning algorithms. In conclusion, the use-case demonstrates the ability to imitate the behavior demonstrated by the engineer through the utilization of knowledge-graphs for the formulation of reward metrics to guide the reinforcement learning-based agent. In contrast to the previous chapter, the current chapter considers actions performed by a single user in training the recommendation agent, although the framework does provide the capability to handle multiple users. The chapter progresses in the following manner:

- Introduction of the considered use-case.
- A review of the application and knowledge extraction methodologies.
- A review of the sources of knowledge in a typical CAD system.
- The introduction of a methodology for the extraction of knowledge from a CAD system.
- The introduction of a novel means for encoding the extracted knowledge.
- The utilization of encoded knowledge in training the imitation agent.

- And finally, plugging the agent back into the application to recommend actions based on user's history.

7.1 Siemens NX Use-case

In order to demonstrate the capability of the developed framework, a use-case of the automation of design decisions in a black box system is considered. The black box system under consideration is Siemens NX, a design tool developed by Siemens PLM [204] that enables the generation of a computational representation of a product through its entire design life-cycle. The application consists of several modules such as CAD for the multi-fidelity geometric modeling of the product, CAE for the representation of the physics experienced by the product during operation, CAM for the manufacturing simulations for NC programming, Technomatix for any discrete event simulations associated with the application, etc. As the current work is rather ambitious in its goal, the work is restricted to the consideration of automation of engineering decisions in the CAD portion of the tool. It is important to note that the framework developed on the other hand, is generic enough to be applicable to other applications as well.

With increasing competition in the engineering tool development market, application service providers are looking to develop more complex systems that are extremely flexible to an extent where the number of decisions or choices that are available to the user at any instant range in the several hundreds, if not thousands. In such a scenario, there is an inherent difficulty for new design engineers in getting acquainted with the application and utilizing it for the purpose at hand. The use-case of Siemens NX is one such application where there is a considerable amount of flexibility in the utilization of the software that a steep learning curve has to be overcome in order to gain

expertise in the software. In addition, it would also require sufficient knowledge about the domain, for example, structural engineering would require different definition of the CAD part in comparison to that of fluid dynamics, in order to satisfy the requirements posed by the design problem. Each of these would take a considerable time to attain and would pose a significant challenge to new engineers who attempt to gain a mastery of the tool. The current established approach for learning the tool is through one of four means, with the last being the most predominant,

1. Interactive paid training sessions in which engineers are educated on the use of the tool on traditional and general purpose problems.
2. Static documentations that function as the user manual for the application.
3. Learning from experts personnel in the organization who have had years of practice in utilization of the product.
4. Learning through experience and mistakes in which better alternatives are identified for existing actions.

An alternative to the above mentioned alternatives that is typically utilized in CAD systems is that of rule-based automation principles [205], [206], but these rule-based (previously known as Knowledge-based Engineering) are predominantly used for decision automation and not recommendations. Further, the document has highlighted the prevalent issues with established approaches relying on rule-based systems for the purpose of context-based usage and this becomes a primary issue when considering the use case of flexible applications such as Siemens NX. But since design is a process of decision making, there has been an investigation into the utilization of Markovian behavior of decisions for the purpose of recommendations [207]–[212]. The frequency-

based probability computation approach would not be suitable for a CAD system in a community setting, as the frequency-based framework fails to account for requirements posed by the design problem in the process of design engineering, i.e., it fails to account for the context under consideration. Further, the frequency-based formulation also fails to account for negative results occurring from the mistakes made by an engineer. These drawbacks encourage the current research work to investigate the utilization of a reward-based Markov chain decision making process in which the user decisions are represented as a Markov chain in which each decision is associated with a reward metric through the use of graph traversals.

In order to test the developed algorithms for the knowledge extraction and representation, several problems are considered. A majority of the problems considered are restricted to single simulations of knowledge extraction from an isolated system, but one use-case of knowledge extraction from multiple different applications working on the same design problem under different requirements is also considered. Finally, an isolated system working once more on realistic use-case of the design of a turbofan engine is considered, where the problem involves the design of an assembly of parts. The use-cases considered as summarized as follows,

- Extraction of knowledge from a drafted cantilever beam, illustrated in Figure 7.1. The designer is tasked with the creation of a CAD model which replicates the model displayed in Figure 7.1 with a given set of parameters. The use-case establishes the ability to extract all the necessary operations performed by an engineer and also the associated parameters associated with these operations.

These are then represented in a knowledge graph to represent the resultant Markov chain.

- The investigation of capture of mistakes and learning from mistakes in an undo, delete, edit scenario. The CAD model here is that of a Brushless Cooling Fan, illustrated in Figure 7.2 in which two versions of the parts are created; one in which the constituent parts are created containing mistakes and the other in which the parts are created using the ideal workflow. The purpose of this exercise is to demonstrate the capability of the framework to, first and foremost, capture back-traces in the design state where the delete or undo or edit operations performed on a part revert it to a previous state and secondly, to demonstrate that the imitation learning algorithm predicts the idealized sequence of operations, i.e., one that does not include the undo or delete operations.
- The generalization of actions across multiple design problems. Here, similar to that of the UAV use-case, the requirements of each of the design problems are encoded with the generated graphical model and utilized in the training process. The CAD model under consideration for this is again the drafted cantilever beam that is illustrated in Figure 7.1, but with different settings for the parameters. As the current work restricts the consideration to recommendation of discrete actions, the purpose of the exercise is to demonstrate the capability to develop a graphical model in which knowledge from several different instances of the application are incorporated.
- The final use case considered is that of an assembly where knowledge from multiple different parts are incorporated into the knowledge graph. For the

purpose of demonstration, the complex system of a turbofan engine is considered. The assembly model of the CAD parts is illustrated in Figure 7.3. The purpose of the exercise is to once more demonstrate the capability of the knowledge extraction framework to capture knowledge from a single instance of the application, but address the representation of different parts in their respective encoded states.

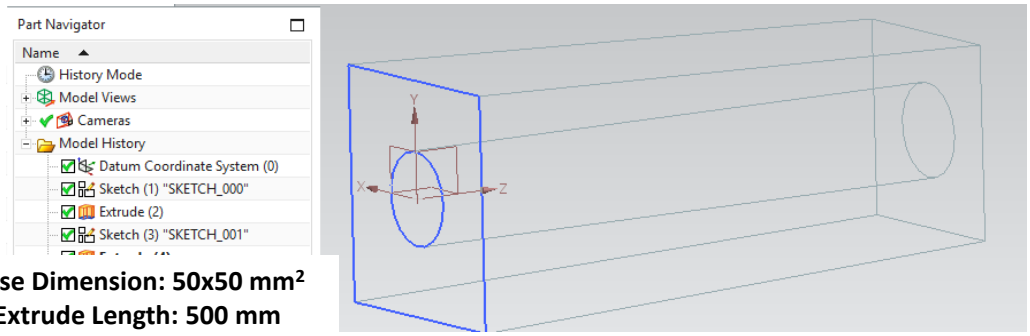


Figure 7.1: A screenshot of the CAD model of a drafted cantilever beam designed with the parameters shown in the image.

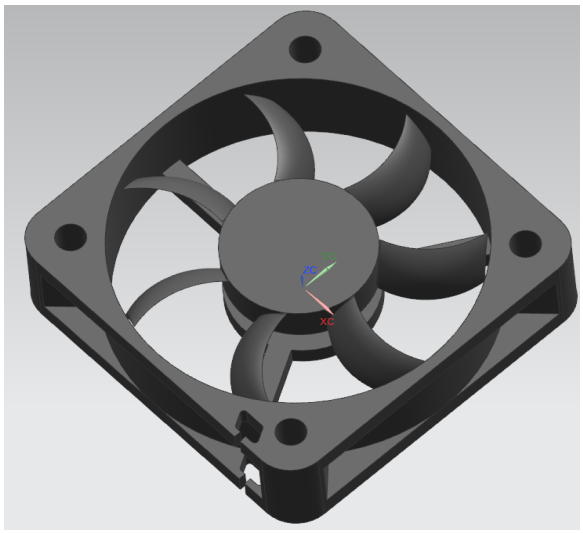


Figure 7.2: A screenshot of the CAD model of Brushless Cooling Fan [213] **modeled to demonstrate the capability to capture reversions in system state.**

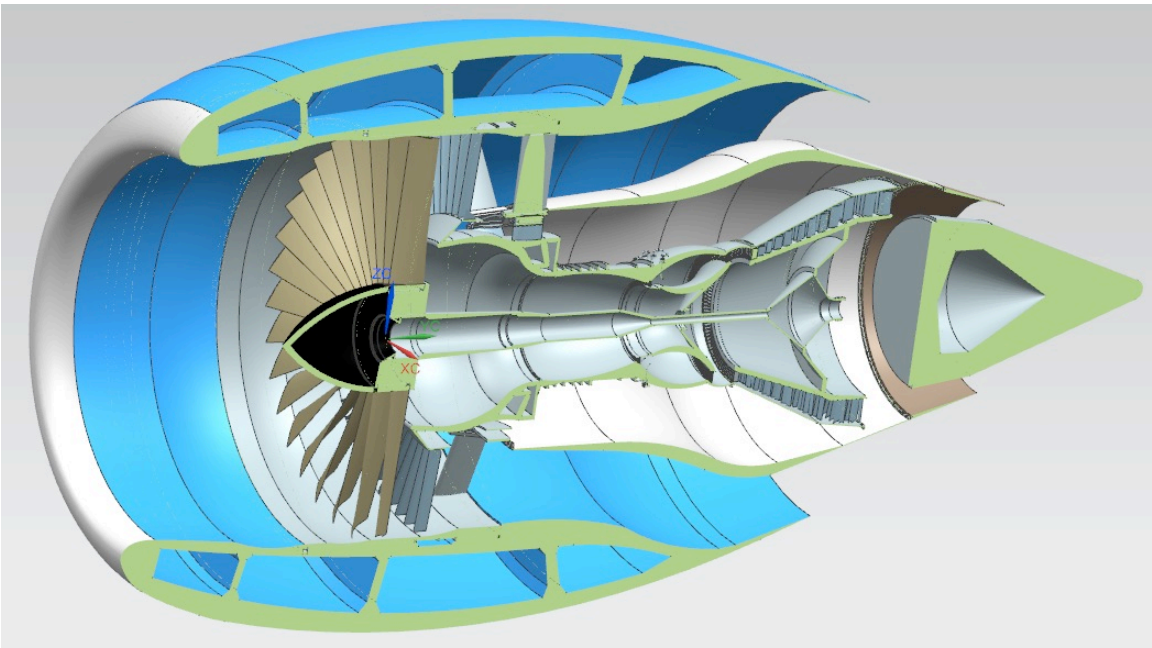


Figure 7.3: The model of the assembly for the Turbofan Engine [214] **and its cross-sectional view showing the components that are CAD part of the assembly.**

7.2 Knowledge Extraction Methodologies in Siemens NX

One of the primary requirements of the research work is the presence of an API from which the knowledge associated with an application can be extracted. Siemens NX meets the requirements through the framework of NXOpen [215], SNAP [216], Knowledge Fusion [205] and GRIP [217] each of which forms a programming paradigm of its own. NXOpen permits the engineer to customize the behavior of any of the NX applications through custom applications or plugins that leverage the open architecture offered by NX. Built on traditional programming paradigm, NXOpen permits the engineer to develop capabilities in several different programming languages such as C++, C#, Java, VB.NET and Python. The Graphical Interactive Programming (GRIP) language of the NX API provides an automation capability for most of the operations within NX through the use of a high-level programming language. While GRIP provides the

capability for automation, it is not suitable for the purpose of knowledge extraction due to its outdated nature [216] and will not be considered in the research work. On the other hand, the Simple NX Application Programming (SNAP) while being relatively simple to use in comparison to NXOpen, is designed for small scale applications, and requires an author license that prevents it from being considered for the reseach work. Finally, Knowledge Fusion (KF) is another interpreted object-oriented programming paradigm that relies on parametric rule-based approaches for the automation of the creation of designs. As this framework relies on the explicit specification of parametric rules, it would be infeasible for the current application as it would fail to automatically adapt to changes in the internal data structure of the application. Thus, NXOpen is chosen as the means for knowledge extraction. As such, there have been four approaches that are identified for the extraction of knowledge from the application. These are,

- User Exits [218]: These are features within NX that permit the execution of a NXOpen application at certain predefined locations (exits) for certain predefined operations, through a pointer based reference to the NXOpen application. Traditionally, user exits have been utilized to replace existing functionality within a certain operation, but these are quite limited in nature as they are defined only for a handful of operations. Thus the utilization of user exits in the current research work is not considered any further, as it fails to provide a generic framework for the extraction of knowledge.
- Menuscripts Handlers [219]: Menuscript is a feature of NXOpen that permits the execution of predefined handler – a NXOpen application – at a certain points of execution of an operation. Traditionally, menuscript handlers function as decorators on the operation executing the requested application either prior to

(PRE) or after the (POST) execution of the operation. While a custom menuscript can be triggered after each operation, NX fails to pass any data related to the operation that triggered the script. Further, there is a necessity to configure each operation of NX to operate with these menuscripts. Given the manual nature of the configuration and the lack of indication of the operation triggering the script handler, menuscripts are dropped from consideration.

- Internal Infinite Tail: A tail is an infinite thread that looks for changes in a certain object typically informing other elements of a running framework of the observed changes. Such systems are typically employed when a continuous observation of an object is sought, as in this case. In the current context, the tail is internal to Siemens NX in order to observe any changes that occur to the CAD part, thus tailing the feature tree of the CAD model. Upon any change the tail would trigger a separate thread that would generate the instantaneous representation of the part file that can then be serialized on the knowledge graph. As there are very few restrictions on the nature of the tail, a modified version of the internal tail is utilized in the current implementation. It is essential to note that Siemens NX only provides the capability to host one or more internal tails, but does not provide an off-the-shelf implementation of it. Thus, in the current implementation, the internal tail is modified to listen to external requests for part state serialization. The external requests are triggered by the external tail as and when user events are identified.
- External Infinite Tail: As with an internal tail, the external tail observes the changes to the state of the Siemens NX application as a black-box from outside the application. The tail attempts to identify user actions by processing one of the

sources of knowledge and by passing a signal to the internal tail (in the form of a lock file) requesting a part serialization. The serialization can then be processed in order to perform the necessary knowledge extraction and representation. As the tail lives completely outside of Siemens NX, these implementations typically require significant processing which may reduce the efficiency of the system.

With the above summarization of the alternatives for knowledge extraction, the current research work develops a methodology in which a combination of the internal and external tail is utilized. The external tail signals to the internal one as and when a change user event occurs. At this point, the internal tail serializes the state of CAD model out to a file which is then processed by the external tail to identify the state of the system and the associated action. These identified state, action and next state tuple are then published to the knowledge graph to form two nodes and an edge between the nodes. The sequence of operations described above is illustrated in Figure 7.4. As the developed framework utilizes Python for the machine learning and data interface operations, a decision is made to utilize NXOpen Python for the external interface between Siemens NX application and the framework, and NXOpen Java as the internal interface that extracts the representation of the parts from Siemens NX.

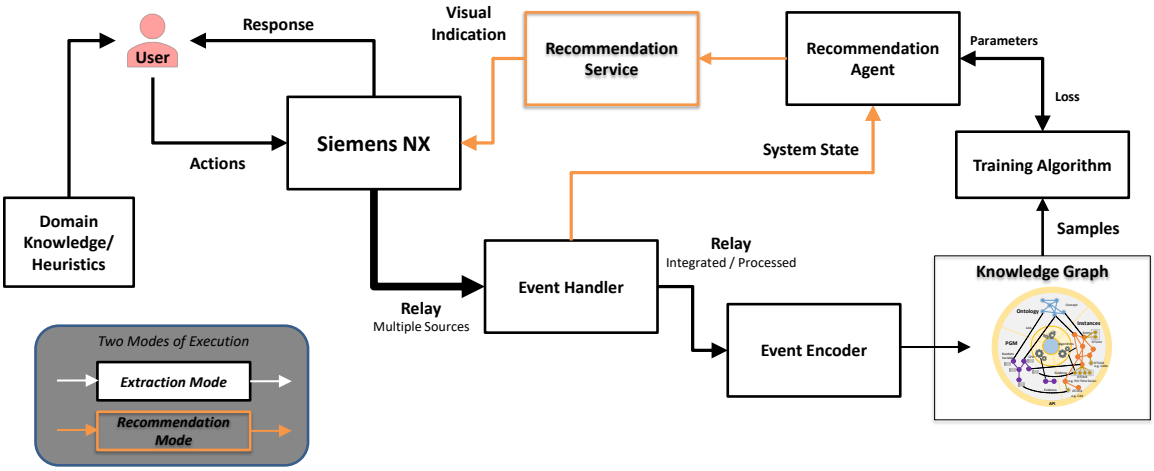


Figure 7.4: A high-level overview of the system architecture in which a knowledge graph is utilized to guide the serialize system states and then train and recommendation of design decision to engineers

7.3 Knowledge Sources in a CAD System

The purpose of a CAD system is the computational representation of the volume of an object that is being modeled, with these representations typically being rendered graphically. While the underlying CAD framework may utilize techniques such as boundary representation [220] or constructive solid geometry [221] for the representation of these volumes, the external user utilizes the graphical display that results from these representations. To be precise, the underlying data structure of the generated model is not visible to the engineer using the black-box CAD system. Thus, even though the system stores a representation of the points, lines, surfaces and bodies and their relationships within the CAD part, such information is only visible to the user in forms of a set of algebraic and mathematical operations on the stored data and relationships. In fact, the CAD framework utilized by most commercial design systems, are a strict trade secret that is not divulged, to an extent that minor changes to the data structure and algorithms in the system can result in significant improvements in processing times and rendering

performances. On the other hand, in order to apply any machine learning technique, one requires the presence of numerical representations of the states of the system. In the Markov decision process context, in addition to the numerical representation of the state of the system, one would also require the generation of a numerical representation for the action.

The current research work looks into alternate means of extraction of these state and actions from sources that are visible to even the engineers using the software. Granted, though most engineers utilize the 3D graphical display as the representation of the system, the current research work does not address the vision-based perception of the CAD system. Instead, the extraction of knowledge from three different sources within the CAD system is considered. CAD systems typically store a parametric representation of the object being designed. Depending on the system being considered these are termed as Expressions (Siemens NX and Autodesk), Parameters (CATIA) or Equations (Solidworks). These store the parametric relationships between the entities in the CAD model in a numeric fashion. While expressions would represent each design uniquely, the numeric values associated with these expressions without an encoding of the CAD model would be meaningless. On the other hand, a representation of the CAD model without the encoding of the expressions would result in a family of designs whose parameters have been abstracted. Thus in order to realize a realistic design, an intrinsic component to the knowledge extracted has to be the expressions stored within the design system. Thus the first source of knowledge forms the expression tables that tabulate the expressions contained in the CAD model. Figure 7.5 illustrates the expression tables across the different CAD applications.

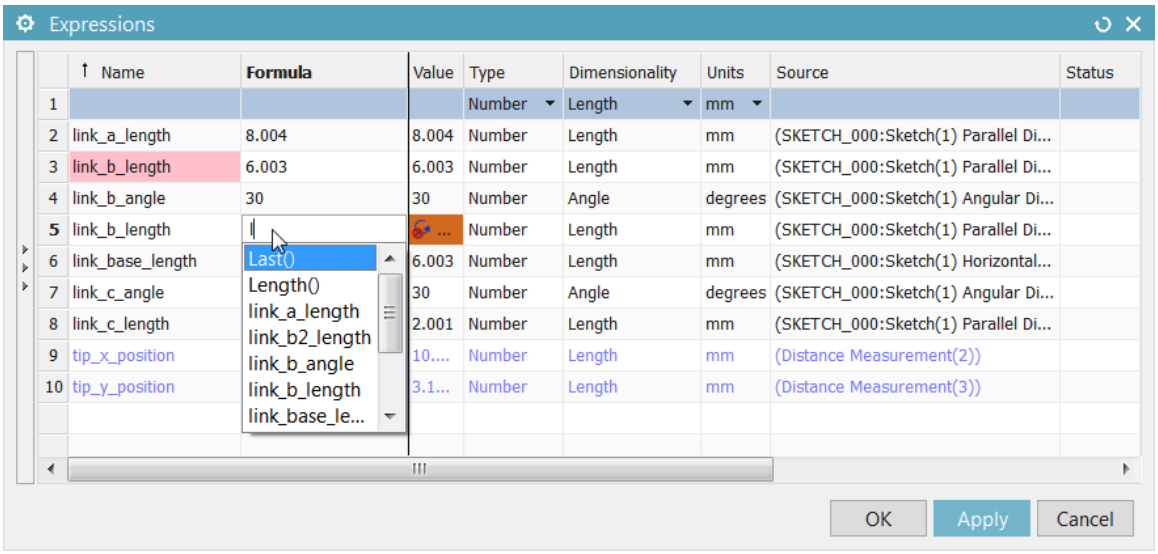


Figure 7.5: Table containing the parameters that defines the instance of a CAD model.

As CAD modeling utilizes a sequential representation of the user's actions, the sequence of operations are typically stored within the system and represented in two forms. First the features that result from the operations performed by the engineer are represented as a Feature tree (Siemens NX), Browser Trees (Autodesk), FeatureManager Design Tree (Solidworks) or Modeling Tree (CATIA), with the contents of each of the tree being the same. The purpose of these trees is to provide the user a visual indication of the CAD entity/feature that was generated as a result of an operation. These trees provide the sequential representation for the different states that the CAD model has observed through the course of changes applied by the engineer. Thus, in order to capture the sequence of states that the CAD model undertakes, the instantaneous state of the feature trees can be utilized to uniquely represent the state of the CAD model. Thus, the feature trees form the second source of knowledge. Figure 7.6 illustrates the feature sequence indicator trees for the different applications.

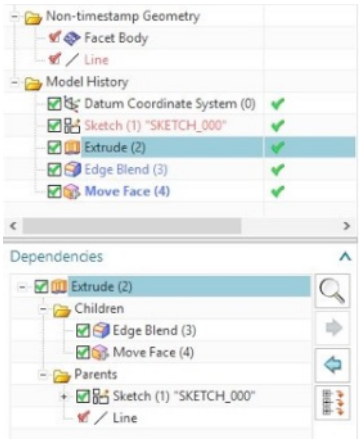


Figure 7.6: Example of a feature tree that represents both the sequence and hierarchy in the CAD model.

Although each entry on the tree uniquely maps to one or a group of operations undertaken to create the entry, in any well-established commercial CAD system there are far too many of operations and possible features to hand engineer the combinations. Thus although the first and second source of knowledge provides a representation for the state of the system, the actions that cause a transition between the states are not encoded within the CAD system. These are typically enforced when the engineer performs a certain action and the instance of the operation is then deleted from memory. In the light of demand for customer support most commercial products log the events that occur with the software application out to a file on disk. These logs would store detailed information about the user's interactions with the application and in the case of Siemens NX it also stores information about the action performed by the user and the parameters associated with the action, including parameters that are hidden from the users in the expression table or the feature trees. Thus, the log files serve as the third and final source of knowledge from which the knowledge graph was developed.

7.3.1 Log Files

The log files of Siemens NX are illustrated in Figure 7.7. The log file provides a detailed source in which each event that occurs in the application is represented in ASCII format. These log files further follow a MACRO based coding style, as illustrated in the figure. These MACROs indicate the occurrence of an event and the block in which the MACRO descriptions is enclosed details the system parameters that the engineer operates with, without exposing the internal activity or computations undertaken. Given that the expression and the feature sources of knowledge can be utilized to completely reconstruct the state of the system, in the current research work, the log files are utilized for two purposes, namely,

- The extraction of the action and the parameters of the action that results from a user action.
- The identification of a change in the state of the system

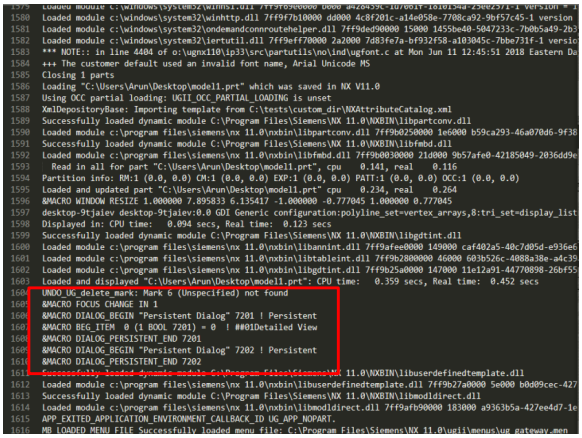


Figure 7.7: An example of the Siemens NX log file with a MACRO section highlighted.

7.3.1.1 Extraction of the user action and the parameter of user action

Given that the log files generated by Siemens NX utilize the MACRO notation for representing events within the application, it is quite straight forward to setup an algorithm that parses text to identify the blocks of the log file that represent the occurrence of an event. In particular, each interaction performed by the engineer is tagged with a particular name, i.e., the MACRO MENU and the termination of the event is again tagged with one of three possibilities, MACRO OK, MACRO APPLY or MACRO CANCEL, with the first two indicating that the engineer has applied the changes made by a certain operation onto the CAD model and the latter indicating that any new change since the most recent one have been disregarded. With these, the identification of blocks of interest is simplified and with each block of interest, is an associated user action. The user action in turn may result in the transition of the system from one state to another, if the operation results in the creation of or edit of existing features. The block of interest is then processed using the regular expressions to identify the parameters and mouse events associated with the user action using the following sequence illustrated in Figure 7.8. Figure 7.8 indicates that given a block of text that has been identified as containing a user action, a block processor executes a regular expression search looking for a set of items on each line of the block. When the entity being search for is identified, the corresponding attribute on the processing object is modified to reflect the item.

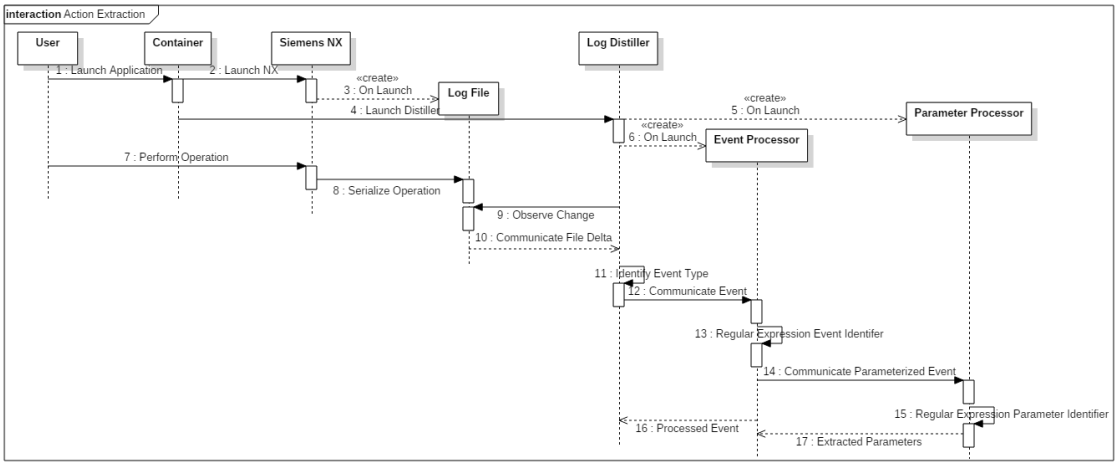


Figure 7.8: Sequence diagram for the processing of the block of text to identify the user action and the parameters associated with the action

7.3.1.2 Identification of an event in the system

The log files created by Siemens NX are typically quite large the processing of which is further hampered by the fact that these log files grow over time for one instance of the application. Further, as the log files store all the information that is generated by the application, it is necessary to process the file in an optimum manner. Finally, the log file rests on the file system and the framework would be decoupled from this file during the operation of the application. Thus, it is necessary to implement a tail on the log file that emits a signal when there are any changes to the file. The changes to the file are a result of the user action in the application and are piped out to the log file in real-time. Thus by having a tail emit a signal indicating the most recent changes that have occurred in the log file, only the pertinent portion of file can be processed severely minimizing the computational effort spent in identifying the user action.

7.3.2 Expression Tables

The expression table of Siemens NX stores the relationship between a parameter that belongs to a certain feature and numerical or mathematical relationship that guides the value of the parameter. Although, expressions within Siemens NX are automatically assigned to features, there are certain features for which the expressions cannot be gathered, for example, the type of operation on an extrude feature, the centre point of operation for a circle, etc. As a result of expressions being automatically assigned, these expressions do not relate to a physical quantity, unless otherwise specified by the engineer. As Siemens NX does not enforce the creation of user defined expressions, it is necessary to process expressions in a more generic manner, in particular, as being associated with the feature and not associated with the CAD model. Siemens NX internally supports multiple different types of expressions, ranging from, strings, booleans, integers, real-valued expressions, 3d coordinates, 3d vectors or a generic list that functions as a container for any of the above. Siemens NX meets the requirements or the assumption for the requirement of a limited amount of API that permits the extraction of information from the application. This is achieved through the use of NXOpen [215]. NXOpen when inspecting the contents of a CAD model permits the identification of expressions associated with any feature within the part. Thus, it is possible to extract all the information from the part file on demand. The extracted information is formatted as illustrated in Figure 7.9, where each expression is associated with a parameter name, an equation, a description, the set of units, and the type of the expression.

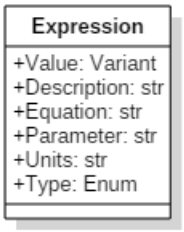


Figure 7.9: The UML diagram associated with the recreated expressions extracted using NXOpen

7.3.3 Feature Trees

The feature tree within Siemens NX is the only manifestation of the sequence of operations performed by the user that is accessible to a user other than the graphical visualization offered by the application. The feature tree indicates a hierarchical relationship between the different features that are created by a certain operations. In this context, a feature could correspond to the datum coordinate system, or sketch that is drawn on a certain plane, or even a solid body operation such as extrude or revolve. The nature of CAD is such that strict parent-children relations between features are enforced internally by the application. These relations prevent the creation of a certain feature prior to the existence of its parent, thus, generating a representation for the changes observed by the system over time. It is to be noted that Siemens NX only displays features that are visible to the engineer in the feature tree, and there are entities associated with most of the feature tree entries that are not visible to the engineer. In fact, it is these entities in most cases that distinguish between two features of the same kind. For example, the feature of datum coordinate system consists of seven entities,

- The origin of the coordinate system.
- Three vectors representing the X, Y and Z axis.

- Three datum planes representing the XY, YZ, and XZ planes.

Thus, the comparison of two coordinate systems entails the comparative analysis of their entities. At this point it is essential to note while a series of parameters can be associated with each feature, and automated extraction schemes can be developed for these, such an approach would be no different from the rule-based systems. In such a setting, if there is any change to the data structure of Siemens NX, then established knowledge extraction approach would very likely fail. Thus, in the current research work, no assumption is made regarding the contents of the entities and the only assumption that is made is regarding the contents of the part file, i.e., the part file are comprised of a sequence of features. The rest of the content of the part file are discovered in an automated fashion. The conceptual result of such a discovery is illustrated in Figure 7.10, where each part designed in the application is shown to be comprised of a state representing the application for the part (in order to enable generalization to applications other than CAD such as CAM and CAE), and a list of features. The features are then decomposed into the constituent expressions and a list of entities, with each entity further being decomposed into a hierarchical set of sub-entities.

The current research work implements a framework in which this decomposition of a CAD model is generated in an automated fashion and is applicable to any generic CAD part.

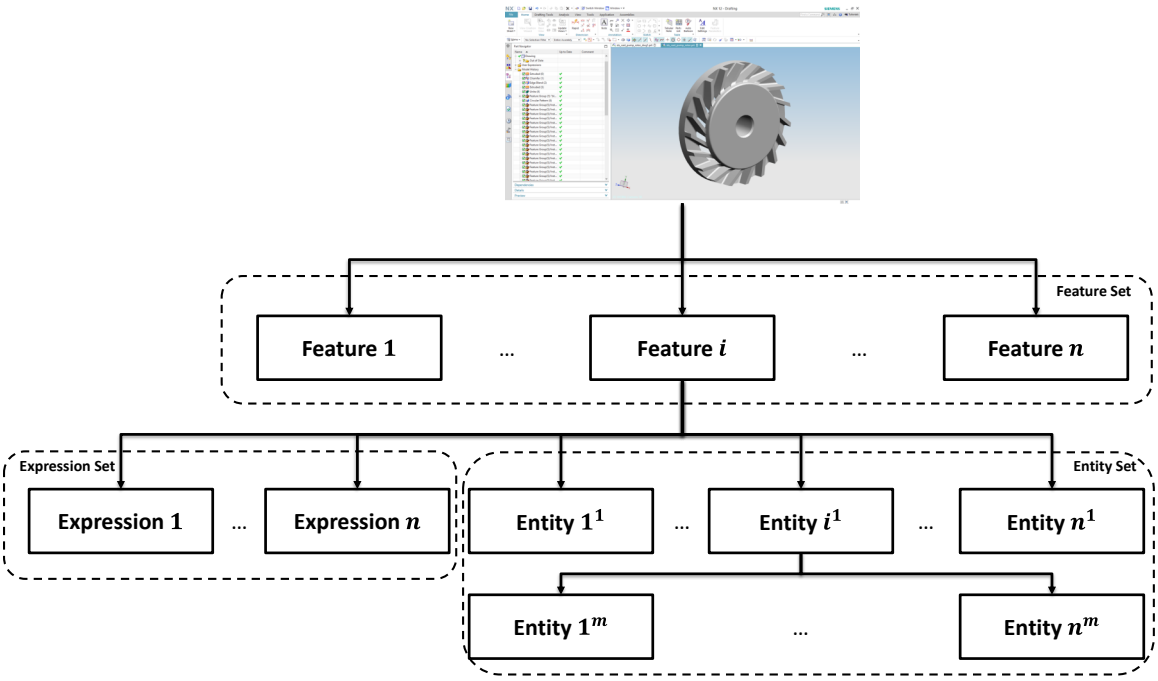


Figure 7.10: A generic hierarchy in the decomposition of a CAD part

7.4 Knowledge Extraction and Representation Mechanism

The current section highlights the details of the implementation behind the knowledge extraction and representation algorithm and highlights the results observed for the different cases considered. In the process of automated knowledge extraction, an object introspection framework is utilized to identify the contents of each part with an initial assumption of the parts being comprised of a set of features. Having generated a tree-based representation for the part file through the execution of the object introspection, an object-based representation scheme is utilized in order to generate a graphical model to capture engineering decisions in the creation of the computational representation of the designed product.

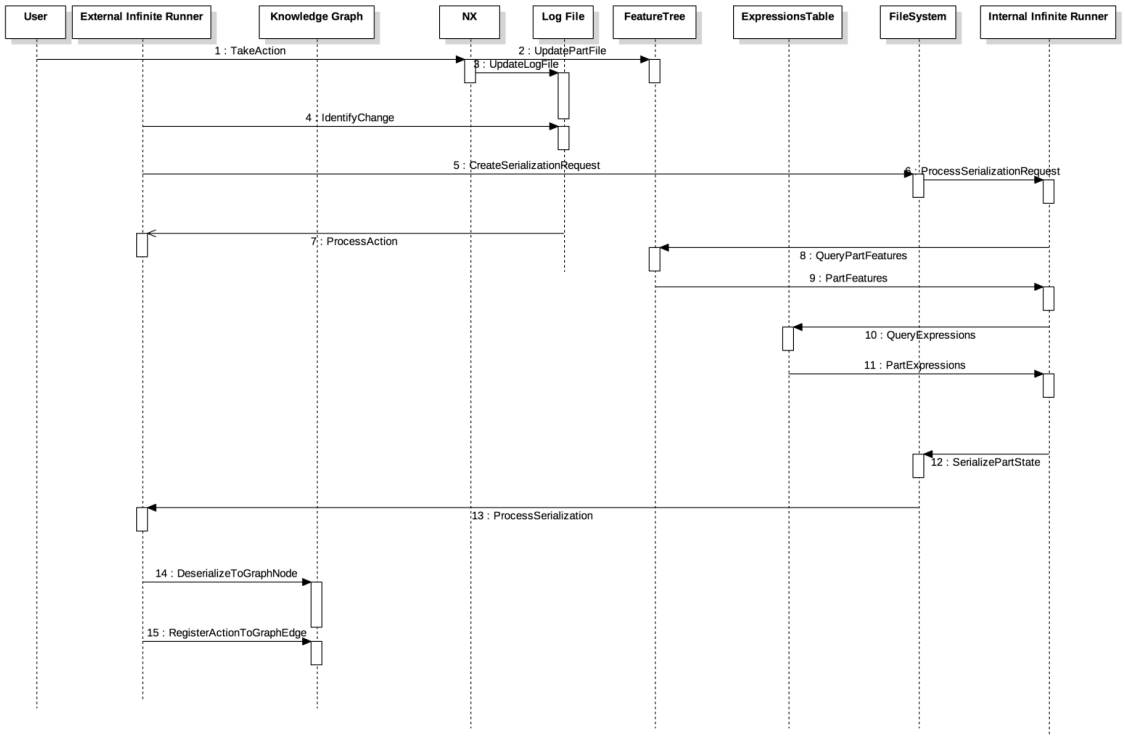


Figure 7.11: Sequence diagram illustrating the extraction of knowledge from Siemens NX

The process of knowledge extraction is illustrated in Figure 7.11. The process of automated knowledge extraction begins with the execution of Siemens NX in an "extraction mode". The process involves an application triggering both an external tail, i.e., a log processor, and the Siemens NX application configured to operate with the external tail. The Siemens NX application is further configured to load an internal tail on start-up which is unloaded only upon the termination of the application. As the log files associated with the instance of NX is created upon launch, the external tails launch of NX as an event gathering the clean state of NX. These states are gathered by creating a lock file that is consumed by the internal tail which then processes the state of the application to output a JSON serialization of the application state. The state JSON is consumed by the framework which regenerates the internal structure of the application in order to

compute differences between previously encoded states of the application. The previously encoded states of the application are retrieved from the knowledge graph (which is hosted on a central server), in order to enable the collaboration between several users. The difference between states are optimized and computed based on object differences, where a difference is computed by evaluating if the objects form isomorphic trees [222]**.** If a pair of isomorphic trees is identified, the resultant state is not added to the knowledge graph but still retained in memory as it would be utilized during the imitation learning process. In parallel, the external tail triggers the processing of the log file in order to gather information pertaining to the user action. All relevant information such as the parameters associated with the action and also the mouse events that are registered by the event are encoded by the action. Utilizing the comparison of the state, if the algorithm identifies the reconstructed states as being identical to ones in the knowledge graph, an object comparison of the action ensues. As the data structure associated with the action is simpler in nature, i.e., an object consisting of two array attributes, the application of tree isomorphic searches is often quicker and easier to conduct returning an indication of if the action between the states are the same. As the framework does not make an assumption about the nature of the Markov decision process, in order to maintain generality, it is assumed that multiple actions from one state can result is transitions to a final resultant state. The algorithm for the knowledge extraction and representation is given below in Algorithm 7.1.

Algorithm1: Knowledge Extraction and Representation Algorithm

Initialize graph server
Initialize graph, $\mathcal{G} = (\mathcal{N}: \phi,\ E: \phi)$ or **Retrieve** graph, $\mathcal{G} = (\mathcal{N}: \{n\}; E: \{e\})$

Initialize $mrn\ = State \equiv$ "$Clean$"
while "Application Is Running"
    If log_file **is** modified

        Identify event in log file
            Process event container, i.e., identify action
                $e_j = process\_event(log\_file)$
            Request serialized state of application
                n_j = Process serialized application state
                $exists =\ is\_isomorphic(n_i,n_j)\ \forall i$
            e_i = edge between node, n_j, and most recent node (mrn).
        If edge, e_j, exists
            $edge\_exists = is\_isomorphic(e_i, e_j)\ \forall\ i$
        If ! $edge\_exists$ and ! $exists$,
            Add new edge, $E = e_j + \{n_i, n_j\}$
        $mrn := n_j$

7.5 Extracted Contents

Algorithm 1 establishes a process for both the creation of a new knowledge graph from a clean state, and also for the utilization and update of an existing knowledge graph containing populated entities. When applied, the algorithm generates a human-readable representation for the state of the CAD model in terms of the features, entities and expressions. The structure of the representation is shown in Figure 7.12 where each node encodes the set of features in a feature container and the name for the state of the system. The name represents the application within Siemens NX that is currently being operated on. The feature container is populated by an array of features that represent the active state of the application (or the CAD part). In the clean state when no part is open, the set of features would be empty.

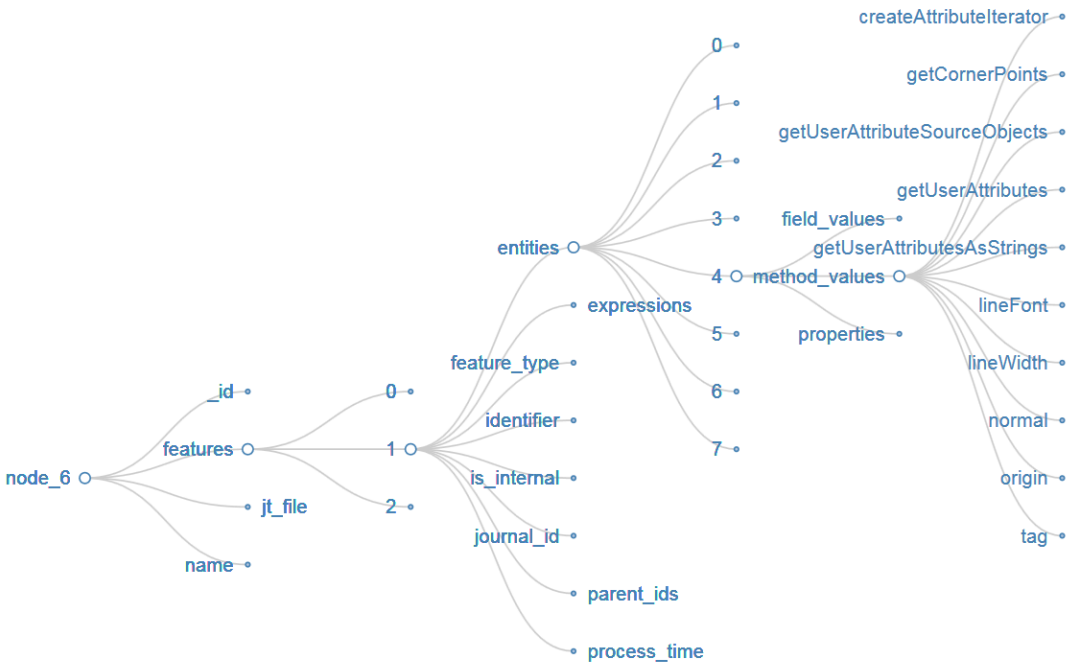


Figure 7.12: The tree-based representation scheme for the contents extracted from each node in the graph

Each feature is decomposed into an array of entities, an array of expressions, the type of the feature, an identifier and the identifiers for the parents feature associated with the feature under consideration. The expressions are composed of an equation, a parameter, its units, its type and the description. The entities on the other hand consist of the attributes and the parameters that uniquely define the feature in addition to the expressions. The structure of the entities is dynamic in nature and would be dependent on the entity under consideration.

7.5.1 Knowledge Extracted in Scenario 1

Figure 7.13 represents the knowledge extracted in the first scenario where the knowledge graph represents the sequence of actions undertaken for the creation of a drafted cantilever beam. The figure demonstrates the progression of the CAD model in

parallel to the generation of the knowledge graph. It has to be observed that the first three nodes form a loop due to an internal transition that occurs with Siemens NX where prior to navigating to the modelling application, Siemens NX passes through a gateway where an initial template part is loaded. The parameters associated with the final part are also indicated in the figure as and when the appropriate nodes are created.

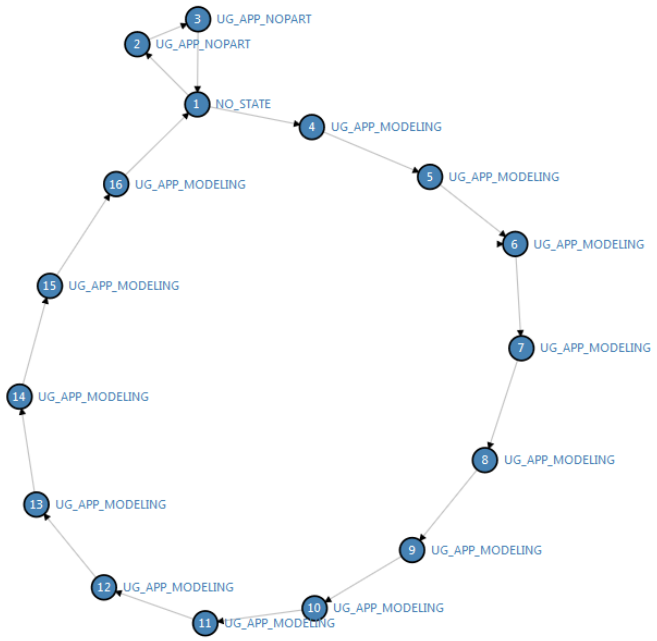


Figure 7.13: Knowledge captured in the creation of the drafted cantilever beam

7.5.2 Knowledge Extracted in Scenario 2

Figure 7.14 represents the knowledge graph that is created in the second scenario. The scenario deals with the creation of a Brushless Cooling Fan model with the goal of generating an appropriate representation of undo, delete and edit operations so as to evaluate the framework's capability to represent back tracking of a systems state as a result of the undo and deletes and a deviation in the system's state as a result of the edit operation. As highlighted by the figure, the knowledge acquisition algorithm is able to

accurately capture the appropriate transitions in states for each of the three operations. The loops identified in the figure could be construed as a mistake on the engineer's part that forces the engineer to trace back their actions. The presence of the loops in one half of the knowledge graph represents the difference between the two created instances of the Brushless Cooling Fan, with the first instance being created with errors and the second one in an ideal manner. This concept plays an important role in the learning process where these loops are identified to associated rewards with the learning process.

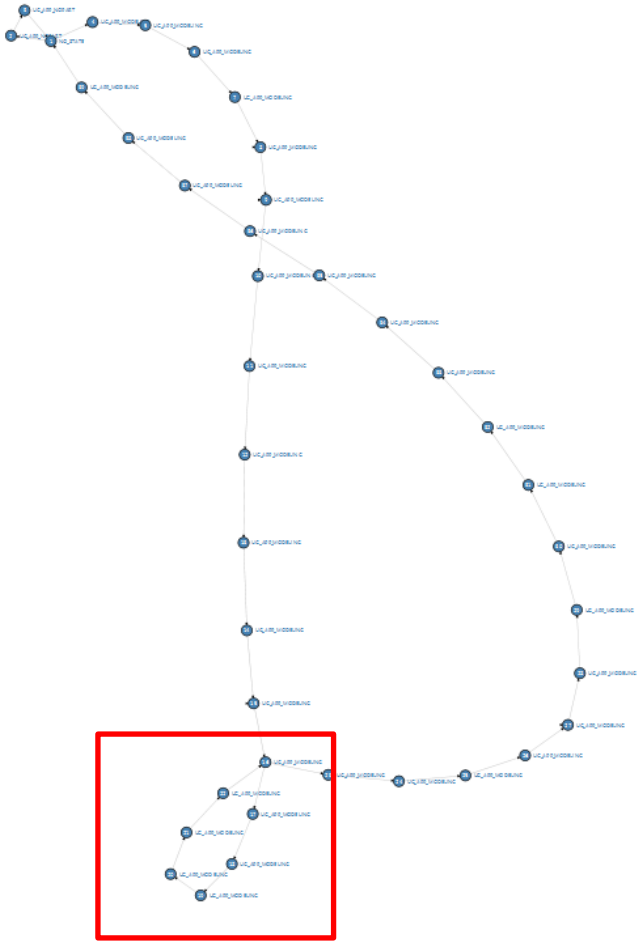


Figure 7.14: Knowledge captured in the creation of a Brushless Cooling Fan with and without errors in the creation of the component

7.5.3 Knowledge Extraction in Scenario 3

The third scenario, illustrated in Figure 7.15, addresses the extraction of knowledge from multiple different sources addressing a common problem with varying requirements. In this scenario, as the requirements drive the parameters associated with the features, the differences in the features are evident with the branching of the knowledge graph, although it is essential to note that each branch of the graph consists of the same set of actions and the same type of nodes. The requirements purely alter the parameter values associated with the nodes. This scenario ensures,

- The ability to distinguish between designs that differ from one another as a result of varying requirements in the knowledge graph.
- In an indirect sense, the ability to incorporate data from multiple instances or multiple different users into a common knowledge graph.

Figure 7.15: Knowledge graph resulting from the modelling of drafted cantilever beams for varying requirements where the loops indicte different requirement.

7.5.4 Knowledge Extraction in Scenario 4

The fourth and final scenario, illustrated in Figure 7.16, address the extraction of knowledge in a realistic scenario of the design of a turbofan engine. The turbofan setting consists of multiple parts as evident from the knowledge graph where the different branches each represent one unique part of the assembly. The scenario functions as a stress test to evaluate the responsivity of the knowledge extraction and representation scheme to the amount of data stored in the graph.

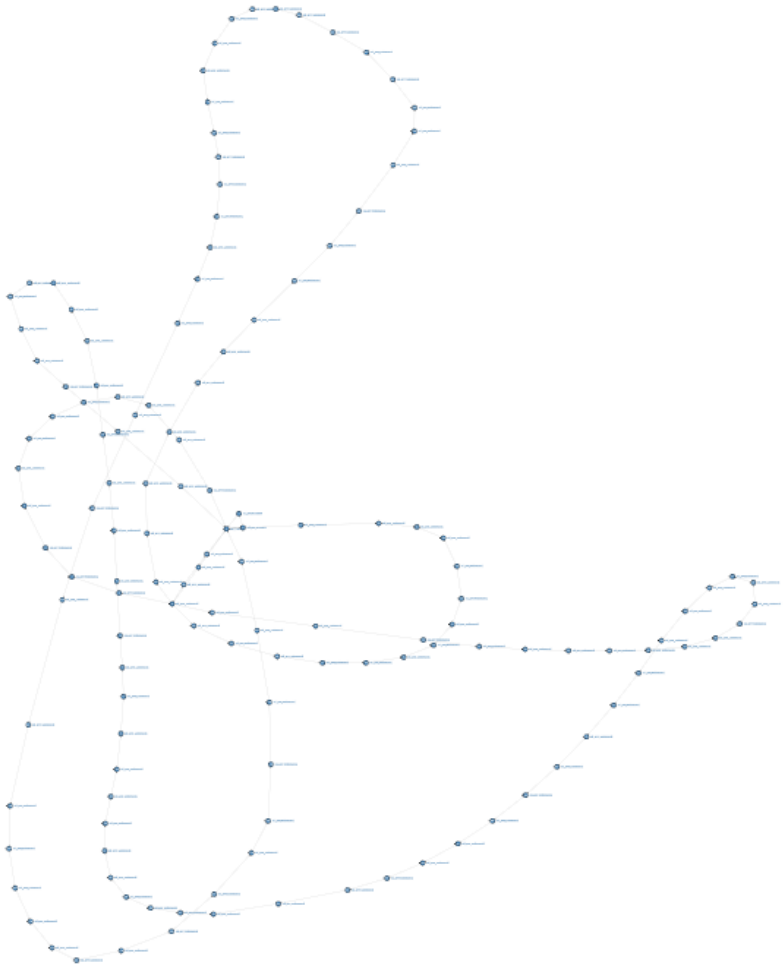


Figure 7.16: Knowledge graph generated from the extraction of knowledge in the process of creating a CAD model of the turbofan engine assembly.

7.6 Knowledge Encoding Mechanism

The current section details the encoding mechanism that is used to generate a vectorized representation for the nodes that have been encoded within Siemens NX. As machine learning algorithms rely on having a numerical representation of the input state of the system in order to enable learning, the current research work develops a generic methodology that can be utilized to vectorize any given CAD part. The approach leverages similarities between the representations of the model and that of a computational graph. An online vectorization scheme is implemented in which state differences are coupled with natural language representations to generate their associated vectorization. Figure 7.17 demonstrates the overall methodology utilized for the vectorization process beginning with the computation of a difference between a state in the knowledge graph and its corresponding abstract representation following which a natural language representation of the differences are generated so as to vectorize these representations using established means.

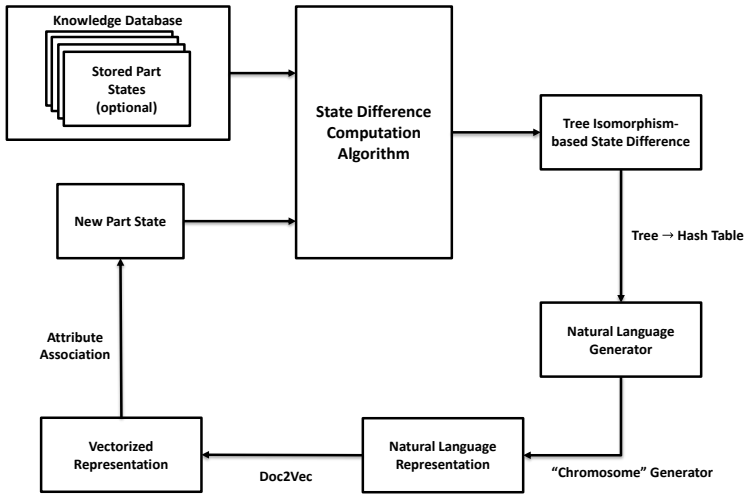


Figure 7.17: Methodology for the generating a vectorized encoding of the knowledge extracted from the CAD model

7.6.1 State differences

The first step of the encoding mechanism is the generation of the state differences. The encoding scheme introduces the new concept of an abstract representation of a graph. The abstraction encodes information about a generic state such it forms the minimum representation of the difference between two states. For example, when two states consisting of datum coordinate systems are considered, the differences between the two states would consist of differences between the origins, the differences between the vectors forming the coordinate axes and finally the differences between the planes forming the coordinate planes. These can in turn be represented as the difference between two 3D coordinates, the differences between three sets of 3D vectors and that between three sets of rotational matrices that represent the orientation of coordinate planes. Thus, the state of the system containing the datum coordinate system can be abstracted in terms of the difference between two instance pairs of the identical feature types that constitute the states. In terms of a mathematical representation, the computed difference would indicate the entities that have changed between two entries of the state such that given an abstract representation and the difference, the new state can be reconstructed through a set addition. The abstraction of the state is in fact represented by a collection of abstract features and abstract entities.

7.6.2 Natural Language Representation

The second step of the encoding mechanism is the generation of a natural language representation for the evaluated state differences. The approach chosen relies on the similarities of the representation of the knowledge extracted and that of the computational graphs. A computational graph (or expression tree) represents the

sequence of operations that are performed and the parameters that are operated on from the leaf of a tree to its root, with the root representing the complete evaluated entity. The framework of gene expression programming [223], [224] proposes an approach in which the expression tree can be converted into a chromosome representation to which a genetic algorithm can be applied to reverse engineer the algebraic equation associated with the expression tree (or computational graph). Given the similarities in the representation of the expression tree and the extracted knowledge, it is demonstrated that a similar approach can be applied to the extracted knowledge to generate a gene expression chromosome. The process is illustrated in Figure 7.18, where the steps involved in the representation of the expression tree (on the left) are utilized to generate an abstract chromosome representation mechanism for a CAD model.

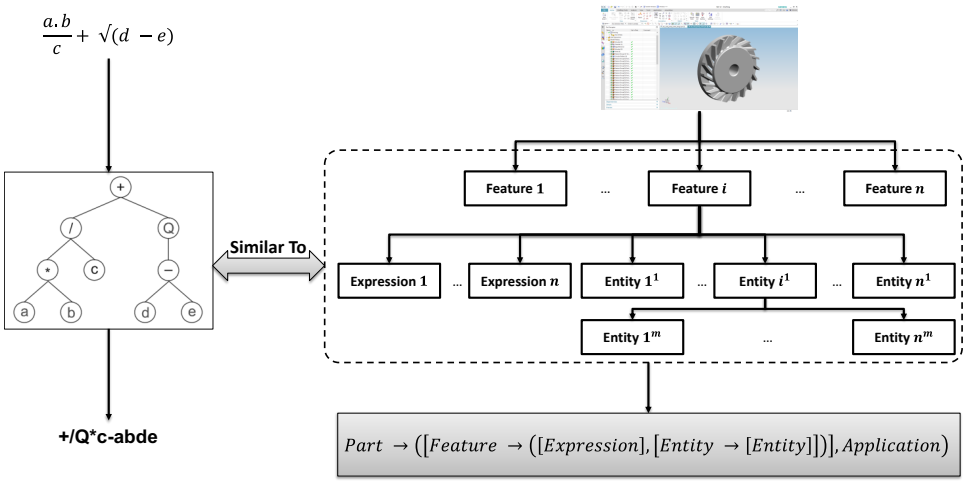


Figure 7.18: Chromosome representation mechanism for a generic CAD model

An inherent issue with the generation of the chromosome representation for the CAD model is the vast amount of data stored within the model. This prevents the application of any online encoding scheme, and thus the proposed chromosome representation scheme is used in conjunction to the state difference computation

mechanism. The state difference scheme generates a hash table representation that cannot be utilized in any type of machine learning algorithm. The contents of the hash table, on the other hand, represent the entities that have been altered from the abstract instance of the state. This knowledge represents the minimum difference in the information about the state that is necessary to uniquely describe the state of the system from a given abstracted baseline. Thus, the differences are utilized to generate a chromosome representation of the state, a result of which is illustrated in Figure 7.19. A key observation that is to be noted is that if and when there is a change to the abstracted representation of the state, the chromosome representation associated with every instance of that state has to be updated to reflect these new changes in the encoding parameters. While this may be computationally expensive in the early phases of the knowledge graph generation, the burden would eventually reduce as the changes to most of the parameters would be captured in the early stages of the knowledge graph. The natural language representation of the parameters would involve the conversion of the hash table to a human-readable ASCII format with formatted syntax and structure to the sentences that is generated.

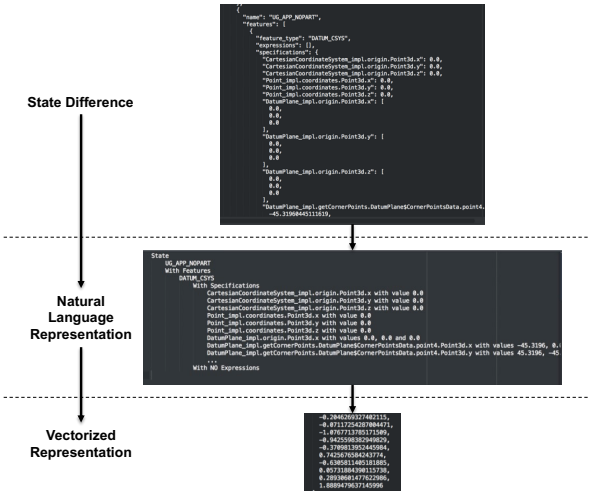


Figure 7.19: An example of a state difference and its associated natural language chromosome representation

7.6.3 Vectorization

The final step in the encoding mechanism is the vectorization of the generated natural language representation for the CAD model. The approach utilizes established techniques in Natural Language Processing [187] in particular, the Doc2Vec [225]. In contrast to the Word2Vec [191] algorithm, Doc2Vec accounts for the relative position of the words in the sentence thereby accounting for the sequences formed by the words in the sentence. The framework generates evaluates three different embedding of 10, 20 and 100 dimensions for the natural language representation. While the higher dimension embedding captures the differences in the generated representation better, they suffer from a generation penalty. This poses a problem for the real-time knowledge graph generation as with every update to the document corpus resulting from a change in the knowledge graph the Doc2Vec model has to be retrained and the embedding generated from the updated model has to be stored in the knowledge graph in real-time.

The primary idea behind the choice of Doc2Vec as the vectorization scheme is the concept that over time, the embeddings generated by the Doc2Vec model would stabilize due to the stabilization of the corpus associated with the Doc2Vec model and by fixing the seed associated with the model, any stochasticity in the vector generation can be eliminated. Figure 7.20 illustrates the sequence of operations and the interactions between the different modules that occur within the implemented framework that dictates the encoding of any given state of the system into a vectorized representation.

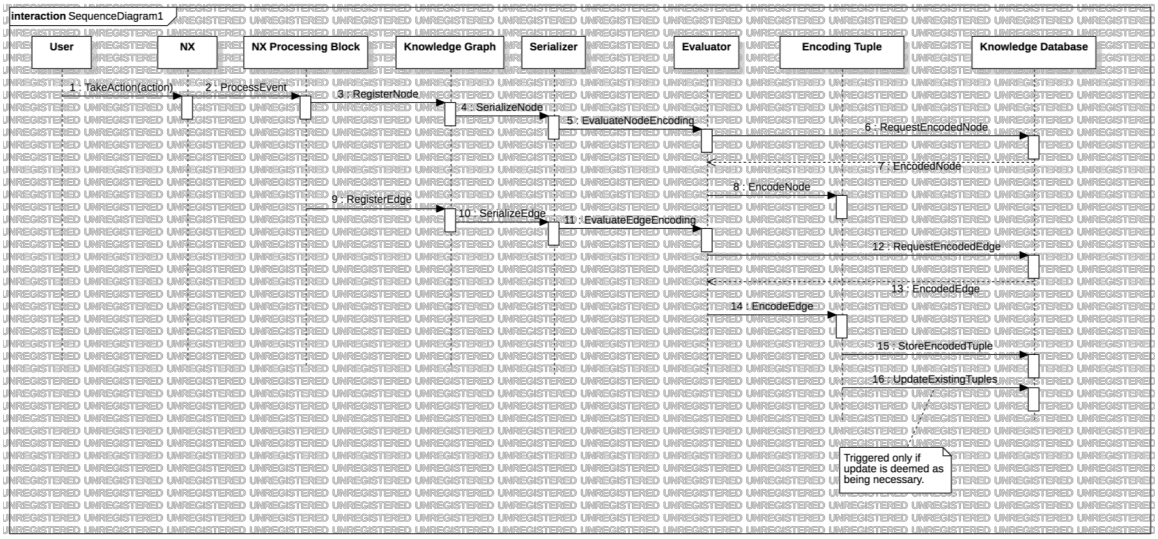
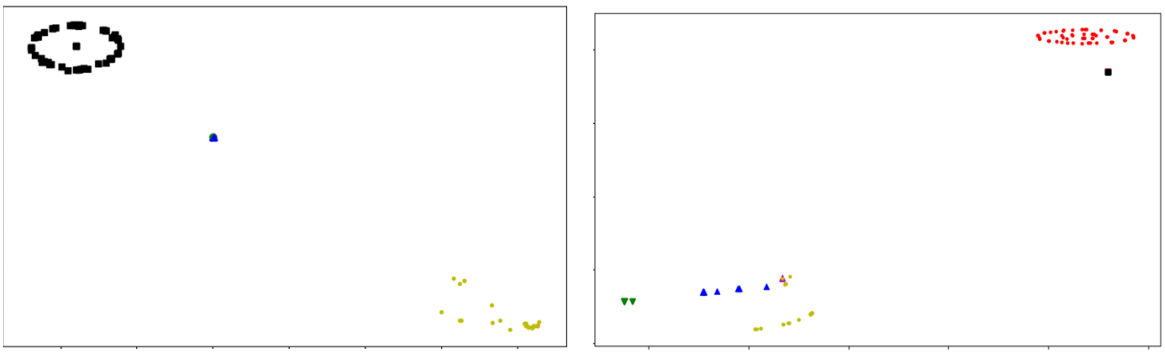


Figure 7.20: Sequence diagram associated with the creation of the vectorized representations for the CAD models

7.6.4 Encoding Results

In order to verify the validity of the knowledge encoding methodology, a visual investigation of the generated embeddings is undertaken. As the methodology hypothesizes that the embedding of the states sharing a common set of features would occupy a common region in space, a visual clustering of the embeddings for a set of seven states (i.e., combinations of features) is undertaken. This is carried out in two ways, first, the embeddings associated with a set of five states generated through a set of 98 events are projected onto a lower dimensional space using the t-SNE [226] algorithm which is illustrated in Figure 7.21 (a) and impact of introduction of additional events and the creation of additional states is evaluated in Figure 7.21 (b). Although the two images use different t-SNE models for the generated embeddings, it is evident from the figures that the embeddings does capture the clusters in the states in an accurate manner. The transformation associated with the two images can be attributed to the change in the document corpus between the two datasets and the use of two different t-SNE models.



(a) (b)

Figure 7.21: A visual evaluation of the validity of the embedding methodology through the identification of distinct clusters for different states of the CAD model

7.7 Knowledge Utilization Mechanism

The final step of the methodology involves the utilization of the gathered knowledge in order to train an artificial agent to replicate the decisions made by the human operator. Owing to the nature of the knowledge extraction, i.e., manual and interactive, the current use case of the research work only addresses the replication of human decisions in the design environment in an optimal manner. The use case does not deal with the transfer of knowledge to other use cases or the generalization of knowledge across requirements. In fact, the current use case limits its consideration to the recommendation of discrete actions, and not the parameters associated with these actions. In order to replicate the actions observed and captured by the knowledge graph, the framework formulates an imitation learning agent that relies on a demonstration-based learning algorithm in order to train the agent in replication of human decisions.

7.7.1 Imitation Learning Algorithm and Agent

The algorithm utilized for this purpose is the Deep Q-Learning from Demonstrations (DQfD) [155]. A “dynamic” deep neural network consisting of a fixed

number of input neurons, a fixed number of hidden layers and neurons but changing number of output neurons is considered. In the current analysis, there are three hidden layers each with a rectified linear activation unit on 64 neurons each is considered. The output layers of the network are linear activated in order to generate an appropriate representation of the value estimates associated with the states and action pair. The network architecture is illustrated in Figure 7.22. The number of neurons in the output layer is dependent upon the number of actions that have been utilized by the engineer and have been encoded in the knowledge graph.

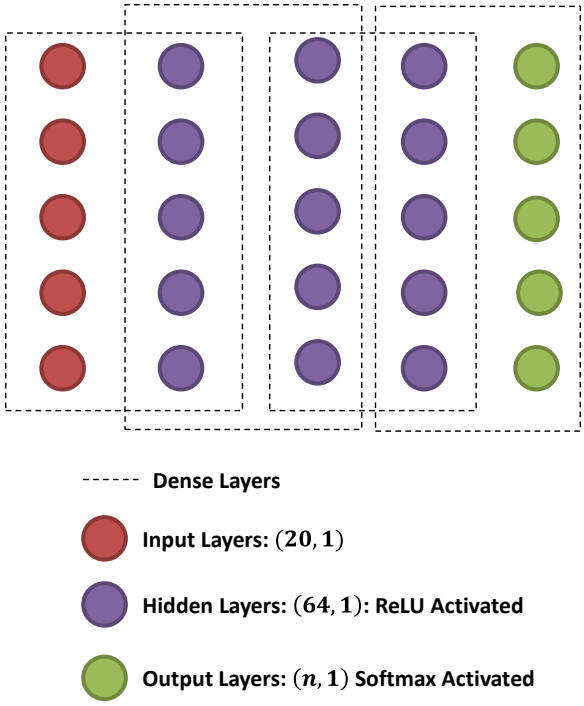


Figure 7.22: The architecture of the neural network utilized for "imitation" learning

Most machine learning algorithms that are prevalent in literature rely on the presence of considerable amount of data in order to extract patterns in the data. While the algorithm of DQfD also relies on the presence of data, improving in performance with the amount of data available, the current workflow artificially simulates the presence of large

quantities of data. This is achieved by intentionally overtraining the neural network on the small quantity of the data in an iterative manner until the errors associated with the replication of the observed data is minimized. This approach ensures that the demonstrated data is replicated perfectly with generalizations across the different instances of the data but without a generalization across the states of the system. But as the target of the exercise is to mimic human decisions, the overfitting strategy proves to be useful in this scenario. In the framework developed, the utilization of the knowledge is triggered by an offline process which generates an artificial agent that can be utilized by the application in the recommendation mode of execution.

7.7.2 Reward and State Formulation

A primary aspect of reinforcement learning is the shaping of the reward function. There has been extensive research into the various different types of reward metrics [48], [227]–[229] and their composition in order to tune the learning algorithm for a particular use case. The current work formulates the rewards in a manner so as to guide the learning algorithm to ignore actions that results in state transitions that reverts the state of the system to a previous state, i.e., paths that lead to actions such as undo, delete, and edits. This is achieved by penalizing these actions with a lower reward than others. As the goal of the learning algorithm is to identify the sequence of actions that result in the largest cumulative reward, this process of penalization would prevent the algorithm from favouring paths with lower rewards. The framework implements a multi-hop graph traversal routine to identify paths that lead to cycles in the knowledge graph. Having identified these cycles, the edges associated with the cycles are attributed with a reward of -2.0, while the edges that do not result in closed cycles are associated with a reward of

-1.0. As the rewards associated with each edge is negative, it automatically transforms the problem formulation to minimizing the number of actions that have to be taken in order to replicate the sequence of operations. In this scenario, such a problem formulation can be viewed as being making the optimal sequence of decisions in the recreation of the CAD model. An example of the formulation of the reward is illustrated in Figure 7.23, while shown graphically the graph traversal algorithm within the framework automatically generates this representation.

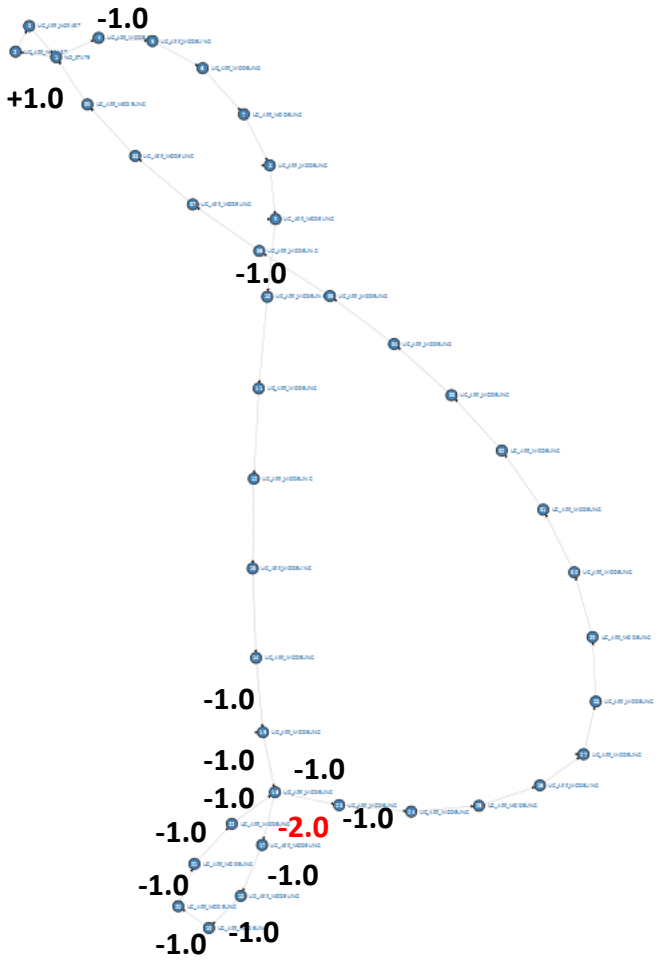


Figure 7.23: An example of the formulated reward for the use case consisting of errors in the creation of the CAD model

Another aspect of the problem formulation in the reinforcement learning domain is the representation of the state of the system. The nature of the CAD application is such that an engineer would arrive at a given state in many different ways. For example, by performing an extrude operation and deleting the resultant feature, the engineer would reach a previous state of the system. This sets of operations can be performed any number of times with varying parameters thus demonstrating that any state in the system can be reached in numerous different ways. In order to account for this issue, it is necessary to capture a history-based state representation. This can be achieved in two ways, one through the use of recurrent neural networks and the other through the use of windowed states. As the recurrent neural network would necessitate a change to the network architecture, a windowed state representation is chosen. The windowed state representation enforces the consideration of the past N states of the system in the creation of the current system state. This is mathematically given as follows,

$$s_i' \epsilon \mathbb{R}^{n,k} = \left[\left[s_j\right]\right] \; \forall \, j \, \epsilon \, \{i, i-1, i-2\, , \ldots, i-k\} \; where \; s_j \epsilon \; \mathbb{R}^{n,1} \qquad [1]$$

In the above equation, the term s_i' represents the modified state of the system that is numerically given by the (n, k) matrix $\left[\left[s_j\right]\right]$ with n dimensional embedded vector and a window size of k. For the current work, while k is used as a parameter of the analysis, in the implementation of the learning algorithm it is kept constant at a value of 5. This would imply that when considering the fourth state of the system in a linear sequence, the mathematical representation of the modified state is given by,

$$s_4' = \left[\left[s_4, s_3, s_2, s_1, NAN\right]\right] when \; k = 5$$

7.8 Recommendation system within Siemens NX

Having trained an artificial agent, the final aspect of the framework is the implementation of the recommendation service within the Siemens NX application. The recommendation service operates in a contextual manner where upon request from the user, the service provides an indication of the action that is to be taken by the user based on the feedback from the artificial agent. The sequence of interactions that occur in the recommendation service is illustrated in Figure 7.24.

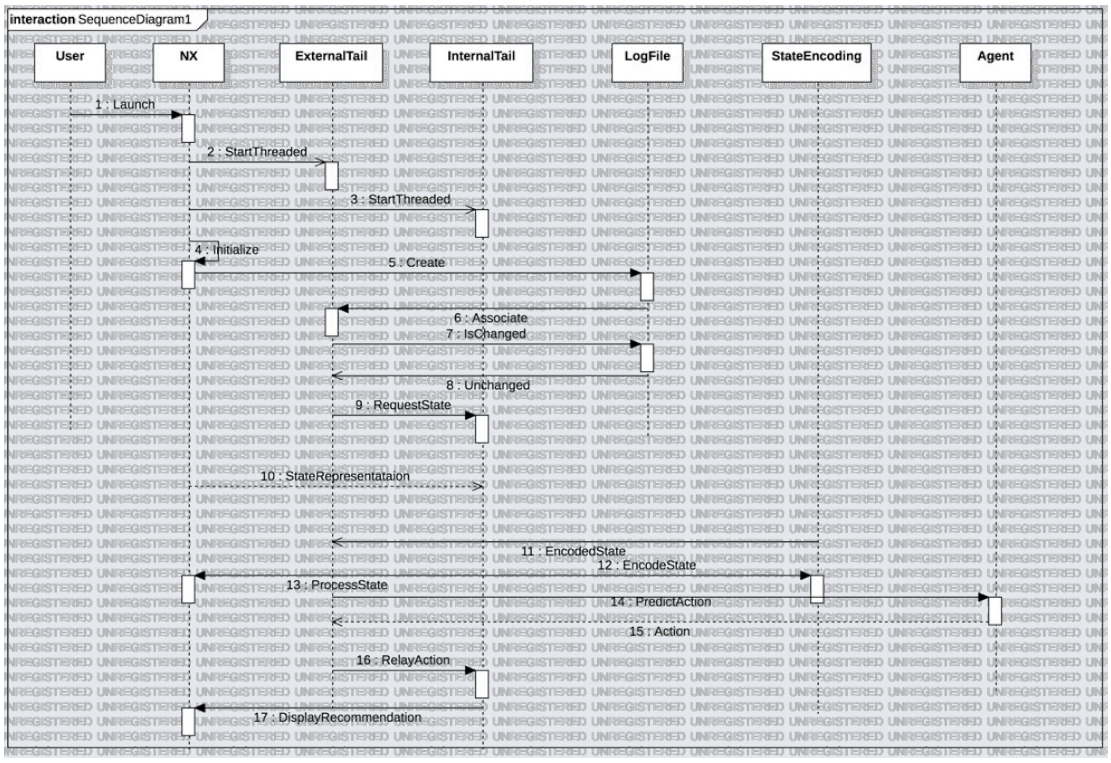


Figure 7.24: The sequence of operations associated with the generation of a recommendation by the imitation learning agent within the Siemens NX application

As with the knowledge extraction workflow, the process beings with an application launcher starting an external tail that manages the interactions with the Siemens NX application and the artificial agent, and also starts the Siemens NX

application. The application once more triggers an internal tail which on demand serializes the state of the system based on which the recommendations can be generated. The engineer triggers a request for the recommendation at which point, the internal tail serializes the state of the system to a file using the previously described knowledge extraction mechanism. The external tail runs on an infinite thread that seeks and consumes the JSON serialization of the part file which when consumed generates the vectorized representation for the part using the previously described mechanism. The vectorized representation is then fed into the imitation learning agent that resides on the server and predicts the operation that is to be performed based on the current and past four states of the application. The generated recommendation is passed onto the external tail which creates a lock file associated with the recommendation that is consumed by the application. The internal tail then consumes the generated recommendation lock file and displays the recommendation within the application. Figure 7.25 demonstrates the integration of the recommendation displays within the Siemens NX application at different stages of the model development.

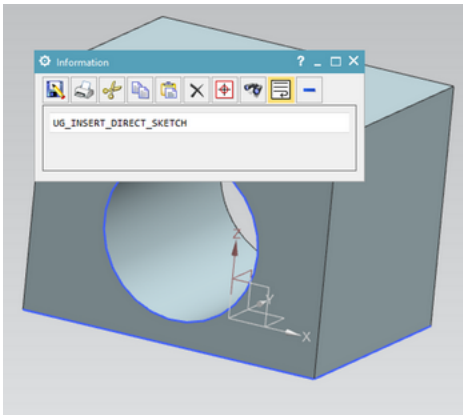


Figure 7.25: Integration of the recommendation service within the Siemens NX application

7.9 Discussion

7.9.1 Achievements of the current work

- The developed process and framework while applied to the application of Siemens NX in truth establishes a process for the vectorization of any generic CAD model. This implies that the processes demonstrated on the Siemens NX application can be transferred to other applications as well. Further, as the methodologies introduced in the chapter are agnostic of the application considered, it can be applied to any design product or assembly.
- The framework demonstrates the utilization of machine learning techniques for the capture of expert knowledge from a black-box system and also the incorporation of the trained models back within the system in order to provide recommendations to engineers on demand. The framework demonstrates the application of imitation learning on extracted data to accurately predict the operations that are to be performed by the engineer in order to retrace the sequence of operations.
- The framework establishes a process for a knowledge graph-based representation of the extracted knowledge that enables the traceability of designs using the stored data. The established processes enable the capture of parameter that reflect different facets of the CAD model and also enables the representation of these on a common manifold for downstream processing.

7.9.2 Limitations of the developed framework

- While the framework addressed the problem of knowledge extraction from a generic application, the developed processes are only demonstrated on CAD. The approaches

developed would be more valuable in the CAE domain as potential benefits associated with utilization of design knowledge and learning from demonstrations are more significant in that field.

- The developed framework relies on the presence of both an API and a log file in order to extract information. While this would be a realistic assumption for most commercial tools, there may be some corner cases where such a capability is not available. Thus the approach would have to be modified in order to account for scenarios that do not permit access to the API or the log files.
- The developed methodologies have focused on the extraction of knowledge from the design process while the design process is being exercised, i.e., extraction of knowledge in an interactive setting. There would have to be modifications made in order to enable the extraction of knowledge from historical parts so as to capture knowledge that already exist in an organization.
- The current approach of generating the embedding of the CAD model through the use of Doc2Vec, represents a one-way transformation of the model which transforms a non-numeric representation to a numeric representation. As the reverse transformation was not necessary for the research work, there were no investigations performed on developing such a capability.

CHAPTER 8. CONCLUSION AND FUTURE WORK

The current research work implements a reinforcement learning framework that relies on the principles of life-long learning in order to assist the engineering design processes. Assistance provided in the form of recommendations of design decisions to the design engineer in the course of utilization of the design environment for a given problem is demonstrated on two separate applications, one involving the utilization of an MBSE application for the purpose of UAV design and the other that involves the creation of CAD models using a commercial black-box application namely Siemens NX. The framework implements aspects of machine learning such as imitation learning from human demonstrations in order to train intelligent agents. The life-long learning aspect of the framework enables adaptation of the trained agents to new and incoming data such that both newly explored portions of the design space and new demonstrations from design engineers are incorporated into the decision making model. The exploratory nature of reinforcement learning algorithm enables the possibility of identifying decision paths that are better that the ones demonstrated by design engineers hence enabling the system to self-learn with the goal of improving the resultant design. An adaptive knowledge graph, representing interactions and effects of human actions, is utilized to encode the sequence of states experienced by the design system with each state represents some unique configuration of a design and an automated approach to the creation of the knowledge graph is implemented through the automation of the knowledge extraction and representation processes. The knowledge is then utilized through the imitation learning process which generates recommendations of actions to design engineers. The approach developed while resulting in evolutionary improvements in the fields of

machine learning, software development and engineering design, leads to a novel approach for an in-product, real-time, contextual recommendation system for the purpose of automation of decisions in complex design systems.

8.1 Summary of Research Questions and Hypotheses

The research work aims develop capabilities that can accelerate the process of design with the aid of expert demonstrations. In order to do so, it is hypothesized that the primary function of each design application is the representation of a state of the design. These design states are driven by knowledge possessed by an engineer and the knowledge is exposed to the design application when the engineer exercises the design process. Thus, the design application can be configured in a manner such that the knowledge exposed to it can be automatically extracted from the system and compiled in a database so as to be utilized by a machine learning algorithm. Further, the machine learning algorithm in its inference mode would enable reproduction of the learning behavior, while generalizing to design scenarios that were not part of the training dataset. Finally, having access to such a machine learning algorithm can enable automated exploration of design spaces in order to find designs that may outperform the training dataset.

The hypothesis leads to the identification of three research questions, each of which is associated with a supporting hypothesis, that are formulated as follows:

8.1.1 Summary of the First Research Question and Hypothesis

Prior to addressing the problem of knowledge extraction and representation of the extracted knowledge, it is essential to identify a means through which the problem of

engineering design can be mathematically represented. The purpose behind this formulation follows the hypothesis that the identification of a mathematical representation enables the generalization of any developed process across design problems. Thus, in order to address this issue of mathematical representation of the problem of engineering design, the characteristics of engineering design are examined reaching a conclusion that the each design problem presents itself as one of sequential decision making. Given that there exist techniques to address the task of sequential decision making in literature, the hypothesis associated with the first research question follows that with the identification of appropriate means of representation and encoding of the knowledge, techniques in solving Markov Decision Processes provides the mathematical formulation for decision making in engineering design problems and that the framework of Reinforcement Learning enables the computational representation of such a problem.

8.1.2 Summary of the Second Research Question and Hypothesis

The second aspect of the research work deals with the automation of knowledge extraction and its representation that makes it amenable to machine learning algorithms. The research question revolves around the identification of the means of identification of knowledge, i.e., what is the knowledge that is to be extracted from a design application, and a means by which to representation the extracted knowledge. The hypothesis associated with this research question builds off of the solutions identified in the research question, i.e., reinforcement learning. Further, the hypothesis leverages the observation that the primary function of design applications is the representation of the design and the change in the state of the design represents the impact of knowledge. Thus, it is

hypothesized that there exists a point in an unknown, possibly infinite, dimensional space that uniquely represents every state of the system, regardless of the feasibility of the design. Further, it is hypothesized that given a context, the unknown dimensional representation can be projected onto a known n-dimensional space, an encoding, that uniquely maps to each observed state of the system. This hypothesis is demonstrated to hold true for two different applications of different complexity over the course of the research work, with specifics of the encoding scheme differing between the two applications.

8.1.3 Summary of the Third Research Question and Hypothesis

The final aspect of the research work conducted is that of enabling the capability of a machine learning algorithm to learn from a mixture of demonstration and experienced data, with the demonstration data being produced from multiple sources. The research question addresses a practical issue where, in a typical engineering setting, data is generated by multiple different engineers each of whom have different levels of expertise in the engineering domain. Thus, the research question addresses the topic of accelerating learning performances in the presence of differing qualities of data. To address this issue, it is hypothesized that an altered version of the prioritized experience replay where in the probability of sampling associated with each sample is altered to account for a "human index". The probability is computed as a function of the expertise associated with the person generating the sample and also that of the error associated with the reproduction of the demonstrated sample. It is demonstrated that the proposed sampling approach significantly improves the performance associated with the

reproduction of demonstrated behaviour in cases having a very large numbers of poor demonstrations.

8.2 Feasibility of the approach

The analysis of the implemented framework is carried out on three fronts. First, it is shown that an agent trained on the problem of UAS design is capable of replicating human-like decisions in the presence of demonstrations. Further, it is shown that if a better decision path is available, the exploratory nature of the algorithm enables the identifications of designs that are better than the best demonstrated one. Finally, an analysis of the robustness of the agent to changes in the set of requirements is performed in order to estimate the flexibility of the framework and its capability to generalize across different but similar problems. A rigorous analysis on the impact of training times, amount of data and the size of the problem is performed in conjunction to the first problem setup. Second, an approach to automate the extraction, representation and utilization of knowledge from multiple sources of information is demonstrated on the problem of automation of engineering systems. Finally, it is shown that the implemented framework outperforms existing state-of-the-art systems that rely on rule-based inference and case-based reasoning. It is shown that the agents trained by the implemented framework are more adaptive to the problem at hand and require fewer configurations in comparison to the state-of-the-art systems.

8.3 Key Contributions of the Research Work

A set of key contribution identified over existing approaches in the field of automation of engineering decisions are summarized as follows:

- The framework introduces a data-driven approach for the utilization of design knowledge in field of engineering. Existing approaches such as that of expert systems rely on hand-crafted knowledge databases that rely on tedious knowledge acquisition processes to generate specialized recommendations as dictated by the application context, making them application specific and difficult to engineer.
- The framework demonstrates the capability of being applicable to multiple different domains and different applications thus, overcoming the shortcomings of the traditional expert system that are hand engineered to one single application domain.
- The framework demonstrates the capability for automated knowledge extraction and representation and the consequent utilization of the knowledge for the purpose of design recommendations and engineering aids. The value of the developed methodology for the automated extraction of knowledge is demonstrated on the Siemens NX use-case where there is a considerable complexity in the design application.
- The utilization of gene expression programming in the generation of a tree-based representation of a CAD model is unique in nature. The problem of vectorization of CAD models is one that is an open topic in literature; as methods that retrieve knowledge from a CAD component do not exist in literature, to the extent of the author's knowledge; and it has been successfully addressed by the current research work through the utilization of natural language processing to convert this tree-based representation to a vectorized representation with an intermediate step of state difference computations.

- The implementation of imitation-learning and demonstration-based learning for the purpose of decision recommendation in the engineering field is once more novel in nature. To the knowledge of the author, there are currently no applications that implement such capabilities and the current work is the first of its kind to apply machine learning for the purpose of engineering design.

8.4 Known Limitations of the Developed Approach

- While the developed methodology for the vectorization of CAD models generates a unique representation of the model based on the contents of the model, it is a one-way transformation. The current approach does not permit the transformation of a point from the n-dimensional embedding space back to the CAD representation. While the current approach has shown to be successful in practice, there are no theoretical guarantees for uniqueness of the transformation routine utilized.
- The framework relies on the presence of an engineer or an automation script that interactively exercises the design process in order to extract knowledge from the application. This implies that in the presence of knowledge that is stored within an organization, in the form of past data, the current framework would not extract any information from such cases. Although the necessary change to enable the extraction of the knowledge from such applications are the next logical extensions of the current research work and are perhaps the lowest hanging fruits from a research perspective, the work in its current form would not be applicable to historical data that has not been processed.

- One of the primary areas of focus in modern deep learning models is the field of transfer learning. While the research touches upon a related topic of generalization of agent's models across multiple different scenarios, it fails to address the topic of transfer of knowledge gained in one use case to another. This would imply that if an agent would need to be applied to a new use-case, a retraining process would need to occur which may be expensive.
- In contrast to the manner in which human perceive data in a design application, i.e., the sense of sight, the current work relies on the availability of a numeric encoding mechanism. This approach prevents the application of end-to-end learning in which a closed loop cycle can be developed to recommend actions based on the computationally enabled human-senses.
- As the framework consists of multiple different modules that have to be deployed in order to trigger the learning on even the most simplest of applications, it can be concluded that the setup costs associated with the framework would be high. This can be attributed to the various different applications that have to be loaded on the server in order and the different tails that have to be developed on the client in order to generate the sever-client communication developed within the current dissertation.

8.5 Directions for Future Work

- **Mathematical investigations of capabilities of deep reinforcement learning.** One of the key topics ignored in the current work is the ability to guarantee performance of the artificial agent. The current research work provides empirical results demonstrating the capabilities of the artificial agent without any

mathematical guarantees to the developed approaches. The topic of mathematical analysis of deep reinforcement learning is an open question in literature.

- **Investigations of alternate neural network architectures and algorithms.** The current research work assumes a constant architecture of the neural network and the algorithms considered in the learning process. Over time there have been new neural network architectures that may be better suited for the purpose investigated in the research work and a comparative analysis of such architectures should be performed. Similarly, new algorithm introduced in the field of deep reinforcement learning would also need to be compared to the ones utilized in the thesis work presented here to ensure the choice of the best possible alternative.
- **Demonstration of the capabilities developed in the thesis work on alternate applications.** The problem of knowledge-based engineer design is one that is prevalent across applications of design to all products. Further, this is a problem that is applicable to all faces of design such as analysis, manufacturing, service etc. Thus, there are a variety of other applications that can benefit from the application of the methods developed in the current work.
- **Investigation of adversarial networks as an alternative to the demonstration based learning framework.** The current research work attempts to view the learning agent and the humans are collaborators working towards the common goal. That is, in the initial stages of learning, the agent relies on demonstrations from the user to improve its performance and in the later stages the agent would be able to recommend designs that are better than that generated by the human. In contrast to this formulation, an adversarial setting can be evaluated, where multiple agents compete against each other or against the human. Literature has

demonstrated the capability of such adversarial networks to generated designs that are considerably different and perhaps better than the set that they were trained on.

- **Incorporation of sequence models to enable end-to-end learning.** The ultimate target of the framework is to enable an end-to-end learning system that is capable of taking an image representation of the design application and recommending or performing an action associated with a problem context. This would require the incorporation of sequence models in order to account for the partial observability associated with the design applications and that of computer vision in order to enable vision-based perception capabilities that are not currently implemented in the framework.

REFERENCES

[1] S. Benson-Amram, B. Dantzer, G. Stricker, E. M. Swanson, and K. E. Holekamp, "Brain size predicts problem-solving ability in mammalian carnivores," *Proc. Natl. Acad. Sci.*, vol. 113, no. 9, p. 2532 LP-2537, Mar. 2016.

[2] G. Roth and U. Dicke, "Evolution of the brain and intelligence," *Trends Cogn. Sci.*, vol. 9, no. 5, pp. 250–257, May 2005.

[3] G. Polya, *Mathematics and Plausible Reasoning: Patterns of Plausible Inference*. Princeton, NJ: Princeton University Press, 1954.

[4] Y. Wang and V. Chiew, "On the cognitive process of human problem solving," *Cogn. Syst. Res.*, vol. 11, no. 1, pp. 81–92, 2010.

[5] E. Hullermeier, "Experience-based decision making: a satisficing decision tree approach," *IEEE Trans. Syst. Man, Cybern. - Part A Syst. Humans*, vol. 35, no. 5, pp. 641–653, Sep. 2005.

[6] P. Carruthers, "Human creativity: Its cognitive basis, its evolution, and its connections with childhood pretence," *Br. J. Philos. Sci.*, 2002.

[7] "A Short History of the Steam Engine," *Nature*, vol. 144, p. 849, Nov. 1939.

[8] F. Nebeker, *Dawn of the Electronic Age: Electrical Technologies in the Shaping of the Modern World, 1914 to 1945*. 2008.

[9] J. Dewar, "The information age and the printing press: looking backward to see ahead," *Ubiquity*. 2000.

[10] T. W. Judd, “A History of Modern Computing,” *Hist. Rev. New Books*, 2000.

[11] P. M. Deane, *The First Industrial Revolution*. Cambridge University Press, 1980.

[12] VDI/VDE-GMA-Fachausschuss „Industrie4.0", “Industrie 4.0 - Auf dem Weg zu einem Referenzmodell,” 2014.

[13] (Accenture) and (GE), “Industrial Internet Insights Report,” *Ind. Insights Rep.*, 2015.

[14] Ian Tattersall, “Homo sapiens,” *Encyclopædia Britannica, inc.* .

[15] “Mind,” *The Oxford English Dictionary*. [Online]. Available: https://en.oxforddictionaries.com/definition/mind.

[16] C. E. Dictionary, “Intelligence,” *Cambridge University Press*, 2018. [Online]. Available: https://dictionary.cambridge.org/us/dictionary/english/intelligence.

[17] Merriam-Webster.com, “Intelligence,” *Merriam-Webster.com*. [Online]. Available: https://www.merriam-webster.com/dictionary/intelligence.

[18] Merriam-Webster.com, “Artificial Intelligence,” *Merriam-Webster.com*, 2018. [Online]. Available: https://www.merriam-webster.com/dictionary/artificial intelligence.

[19] P. H. Winston, *Artificial Intelligence - 3rd Edition*. 1992.

[20] C. E. Dictionary, “Thinking,” *Cambridge University Press*, 2018. [Online]. Available: https://dictionary.cambridge.org/us/dictionary/english/thinking.

[21] C. E. Dictionary, “Understanding,” *Cambridge University Press*, 2018. [Online]. Available: https://dictionary.cambridge.org/us/dictionary/english/understanding.

[22] S. French, J. Maule, and N. Papamichail, *Decision behaviour, analysis and support*. 2009.

[23] J. L. Croissant, “Tacit and explicit knowledge,” *Choice Curr. Rev. Acad. Libr.*, 2011.

[24] M. A. Boden, *Artificial intelligence and natural man / Margaret A. Boden*, 2nd ed., E. Basic Books New York, 1987.

[25] M. Minsky, *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. 2006.

[26] A. Tate, “Planning,” *The MIT Encyclopedia of Cognitive Sciences*. MIT Press, 1999.

[27] D. Riesberg, “Learning,” *The MIT Encyclopedia of Cognitive Sciences*. MIT Press, 1999.

[28] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, 1943.

[29] D. O. ; E. A. Hebb and T. J. Bussey, “The Organization of Behavior,” 1949.

[30] C. Smith, B. McGuire, T. Huang, and G. Yang, “The History of Artificial Intelligence,” *Hist. Artif. Intell.*, 2006.

[31] J. McCarthy, “Recursive functions symbolic expressions and their computation by

machine, Part I," *Commun. ACM*, 1960.

[32] M. Minsky, "Steps toward Artificial Intelligence," *Proc. IRE*, 1961.

[33] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, 1971.

[34] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, 1958.

[35] E. H. Shortliffe, D. a. B. Lindberg, and E.H.S., *Computer-Based Medical Consultations: Mycin*. 1976.

[36] R. Duda, J. Gaschnig, and P. Hart, "MODEL DESIGN IN THE PROSPECTOR CONSULTANT SYSTEM FOR MINERAL EXPLORATION," *Readings Artif. Intell.*, 1981.

[37] A. N. Campbell, V. F. Hollister, R. O. Duda, and P. E. Hart, "Recognition of a hidden mineral deposit by an artificial intelligence program," *Science (80-.).*, 1982.

[38] A. Challinor, "Expert systems," in *Applied Agrometeorology*, 2010.

[39] A. N. Ramesh, C. Kambhampati, J. R. T. Monson, and P. J. Drew, "Artificial intelligence in medicine," *Annals of the Royal College of Surgeons of England*. 2004.

[40] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[41] B. Speelpenning, “Compiling fast partial derivatives of functions given by algorithms.”

[42] G. Tesauro, “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” *Neural Comput.*, 1994.

[43] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*. 2015.

[44] Y. Bengio, “Learning Deep Architectures for AI,” *Found. Trends® Mach. Learn.*, 2009.

[45] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*. 2015.

[46] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*. 2017.

[47] C. Finn and S. Levine, “Deep visual foresight for planning robot motion,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[48] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[49] M. Gell-Mann, “What is complexity,” *Complexity*, 1995.

[50] M. Mitchell, *Complexity a guided tour*. 2013.

[51] S. Lloyd, “Measures of Complexity: A Nonexhaustive List,” *IEEE Control Syst.*, 2001.

[52] J. Ladyman, J. Lambert, and K. Wiesner, “What is a complex system?,” *European Journal for Philosophy of Science*. 2013.

[53] H. a Simon, “The Architecture of Complexity: Hierarchic Systems,” *Sci. Artif.*, 1962.

[54] M. Uflacker and D. Busse, “Complexity in Enterprise Applications vs. Simplicity in User Experience,” in *Human-Computer Interaction. HCI Applications and Services, Lecture Notes in Computer Science*, 2007.

[55] ANSYS Inc., “ANSYS, Inc. Fourth Quarter and Fiscal Year 2016 Earnings Announcement,” 2017.

[56] Siemens, “Annual Report 2017,” 2018.

[57] D. Systemes, “2017 3D Experience Annual Report,” 2018.

[58] MathWorks, “Company Overview,” *MathWorks*, 2018. [Online]. Available: https://www.mathworks.com/company/aboutus.html?s_cid=wiki_mathworks_1.

[59] G. A. Hazelrigg, “A Framework for Decision-Based Engineering Design,” *J. Mech. Des.*, 1998.

[60] R. Cubitt, J. W. Payne, J. R. Bettman, E. J. Johnson, and Y. B. Choi, “The Adaptive Decision Maker.,” *Econ. J.*, 1995.

[61] G. A. Gorry and M. S. Morton, “A framework for management information systems,” *Sloan management review*. 1971.

[62] G. Klein, “A Recognition-Primed Decision (RPD) Model of Rapid Decision

Making," in *Decision making in action: Models and methods*, 1993.

[63] D. Snowden, "Complex acts of knowing: Paradox and descriptive self-awareness," *J. Knowl. Manag.*, 2002.

[64] H. A. Simon, "Bounded Rationality," in *Utility and Probability*, 1990.

[65] H. A. Simon, "Rationality as process and as product of thought," in *American Economic Review*, 1978.

[66] P. J. H. King, "Decision tables," *Comput. J.*, 1967.

[67] L. Savage, "The foundations of statistics reconsidered," *Proc. Fourth Berkeley Symp. Math. Stat. Probab. Vol. 1*, 1961.

[68] L. Rokach and O. Maimon, *Data mining with decision trees : theroy and applications*. 2008.

[69] R. Howard and J. Matheson, "Influence diagrams," *Decis. Anal. 2 (3), Sept. 2005 (reprinted).*, 1981.

[70] D. K. Owens, R. D. Shachter, and R. F. Nease, "Representation and analysis of medical decision problems with influence diagrams," *Med. Decis. Mak.*, 1997.

[71] H. Mintzberg, D. Raisinghani, and A. Theoret, "The Structure of 'Unstructured' Decision Processes," *Adm. Sci. Q.*, 1976.

[72] H. A. Simon, "Theories of Decision-Making in Economics and Behavioral Science," *Am. Econ. Rev.*, 1959.

[73] Y. Li, “An Intelligent, Knowledge-based Multiple Criteria Decision Making Advisor for Systems Design,” *Decis. Support Syst.*, 2007.

[74] G. A. Hazelrigg, *FUNDAMENTALS OF DECISION MAKING : FOR ENGINEERING DESIGN*. NEILS CORP, 2012.

[75] R. A. Howard, “Decision Analysis: Practice and Promise,” *Manage. Sci.*, 1988.

[76] M. D. RYCHENER, “Expert systems for engineering design,” *Expert Syst.*, 1985.

[77] F. G. Filip, “Decision support and control for large-scale complex systems,” in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 2007.

[78] McLeod Raymond Jr., *Information Systems*. New York, NY, USA: Macmillan Publishing Company., 1990.

[79] D. J. Power, “A Brief History of Decision Support Systems,” *Decis. Support Syst.*, 2007.

[80] D. J. Power, “Concepts and Resources for Managers,” *Decis. Support Syst.*, 2002.

[81] D. A. Waterman, *A Guide to Expert Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1985.

[82] R. H. G. Field, “Crucial Decisions: Leadership in Policy-making and Crisis Management.,” *Acad. Manag. Rev.*, 1990.

[83] K. Young, “Mimicking Fraudsters,” *The Guardian*, 2004. .

[84] U. Flemming, “Rule-based systems in computer-aided architectural design,”

Expert Syst. Eng. Des., pp. 93–112, 1988.

[85] E. A. Feigenbaum, "Expert Systems: Principles and Practice," *Encycl. Comput. Sci. Engneering*, 1992.

[86] K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer, "The role of deliberate practice in the acquisition of expert performance.," *Psychol. Rev.*, 1993.

[87] K. A. Ericsson and N. Charness, "'Expert performance: Its structure and acquisition': Reply.," *Am. Psychol.*, 1995.

[88] R. A. Cutietta and B. S. Bloom, "Developing Talent in Young People," *Music Educ. J.*, 1985.

[89] E. Raskin, "Comparison of scientific and literary ability: a biographical study of eminent scientists and men of letters of the nineteenth century," *J. Abnorm. Soc. Psychol.*, 1936.

[90] J. Heatherton, "An Introduction to Expert Systems and Knowledge Acquisition Techniques," 1990.

[91] M. Negnevitsky, *Artificial intelligence: A guide to intelligent systems*. 2011.

[92] B. S. Todd, *An Introduction to Expert Systems*. Oxford University Computing Laboratory, Programming Research Group, 1992.

[93] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. 2003.

[94] Y. Shira and F. R. D. Tsujii, "Artifical intelligence: Concepts, techniques, and applications," 2018.

[95] D. Barber, *Bayesian Reasoning and Machine Learning*. New York, NY, USA: Cambridge University Press, 2012.

[96] E. H. Shortliffe and B. G. Buchanan, "A model of inexact reasoning in medicine," *Math. Biosci.*, 1975.

[97] L. A. Zadeh, "Fuzzy sets," *Inf. Control*, 1965.

[98] S. Gottwald, "Many-Valued Logics," in *Philosophy of Logic*, 2007.

[99] E. Turban, J. E. Aronson, and T.-P. Liang, "Decision Support Systems and Business Intelligence," *Decis. Support Bus. Intell. Syst. 7/E*, 2007.

[100] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man. Mach. Stud.*, 1975.

[101] E. Cox, *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems, Volume 1*. 1994.

[102] M. Sugeno, *Industrial Applications of Fuzzy Control*. New York, NY, USA: Elsevier Science Inc., 1985.

[103] M. Minsky, "A framework for representing knowledge," in *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, 2013.

[104] R. Lopez de Mantaras, P. Cunningham, and P. Perner, "Emergent case-based reasoning applications," *Knowledge Engineering Review*. 2005.

[105] J. L. Arcos and R. L. De Mántaras, "An interactive case-based reasoning approach for generating expressive music," *Appl. Intell.*, 2001.

[106] J. E. Laird, “Using a computer game to develop advanced Al,” *Computer (Long. Beach. Calif).*, 2001.

[107] I. Bichindaritz and C. Marling, “Case-based Reasoning in the Health Sciences: What’s Next?,” *Artif. Intell. Med.*, vol. 36, no. 2, pp. 127–135, Feb. 2006.

[108] C. Fulong, W. Chao, Z. Hong, Z. Bo, and W. Fan, “SAR images classification using case-based reasoning method,” in *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2007.

[109] M. M. Richter and A. Aamodt, “Case-based reasoning foundations,” *Knowl. Eng. Rev.*, 2005.

[110] R. Bergmann, J. Kolodner, and E. Plaza, “Representation in case-based reasoning,” *Knowledge Engineering Review*. 2005.

[111] B. W. Porter, R. Bareiss, and R. C. Holte, “Concept learning and heuristic classification in weak-theory domains,” *Artif. Intell.*, 1990.

[112] E. Plaza, “Cases as terms: A feature term approach to the structured representation of cases,” in *Case-Based Reasoning Research and Development*, 1995, pp. 265–276.

[113] R. Bergmann, *Experience Management: Foundations, Development Methodology, and Internet-based Applications*. Berlin, Heidelberg: Springer-Verlag, 2002.

[114] M. Lenz and H. D. Burkhard, “Case retrieval nets: Basic ideas and extensions,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996.

[115] M. M. Richter, “Knowledge Containers,” *Readings Case-Based Reason.*, 2003.

[116] D. B. Leake, “CBR in Context: The Present and Future,” in *Case-based Reasoning: Experiences, Lessons, and Future Directions*, 1996.

[117] C. Larman and V. R. Basili, “Iterative and incremental development: A brief history,” *Computer*. 2003.

[118] I. Kameny, *Guide for the management of expert systems development*. Rand, 1989.

[119] G. GUIDA and C. TASSO, “Building Expert Systems: From Life Cycle to Development Methodology,” in *Topics in Expert System Design*, vol. 5, G. GUIDA and C. TASSO, Eds. North-Holland, 1989, pp. 3–24.

[120] L. Millette, “Improving the knowledge-based expert system lifecycle,” 2012.

[121] M. Welbank, “An overview of knowledge acquisition methods,” *Interact. Comput.*, 1990.

[122] J. Cullen and A. Bryman, “The Knowledge Acquisition Bottleneck: Time for Reassessment?,” *Expert Syst.*, 1988.

[123] J. Hawkins, “On Intelligence,” *Book*, 2005.

[124] J. Glascher *et al.*, “Lesion mapping of cognitive control and value-based decision making in the prefrontal cortex,” *Proc. Natl. Acad. Sci.*, 2012.

[125] P. Ralph and Y. Wand, “A proposal for a formal definition of the design concept,” in *Lecture Notes in Business Information Processing*, 2009.

[126] ABET, “Criteria for Accrediting Engineering Programs, 2018 – 2019,” *ABET*, 2018. [Online]. Available: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2018-2019/.

[127] S. Khandani, “Engineering Design Process,” 2005.

[128] D. L. Householder and C. E. Hailey, “Incorporating Engineering Design Challenges into STEM Courses,” 2012.

[129] G. Dieter and L. Schmidt, *Engineering Design*. McGraw-Hill Education, 2009.

[130] M. L. Littman, “Algorithms for Sequential Decision-making,” Brown University, Providence, RI, USA, 1996.

[131] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[132] R. A. Howard, *Dynamic Programming and Markov Processes*. Published jointly by the Technology Press of the Massachusetts Institute of Technology and, 1960.

[133] R. Bellman, “The theory of dynamic programming,” *Bull. Amer. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 1954.

[134] C. Boutilier, T. Dean, and S. Hanks, “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage,” *J. Artif. Intell. Res.*, 1999.

[135] M. Van Otterlo and M. Wiering, “Reinforcement learning and markov decision processes,” in *Adaptation, Learning, and Optimization*, 2012.

[136] P. C. Fishburn, “Utility Theory for Decision Making,” *Res. Anal. Corp.*, 1970.

[137] A. M. Andrew, “Reinforcement Learning: An Introduction,” *Kybernetes*. 1998.

[138] A. G. Barto and T. G. Dietterich, “Reinforcement learning and its relationship to supervised learning,” in *Handbook of Learning and Approximate Dynamic Programming*, 2004.

[139] A. G. Kohls, “Reinforcement learning-based multi-agent system for network traffic signal control,” *IET Intell. Transp. Syst.*, vol. 4, no. 2, p. 128–135(7), Jun. 2010.

[140] M. Abdoos, N. Mozayani, and A. L. C. Bazzan, “Holonic Multi-agent System for Traffic Signals Control,” *Eng. Appl. Artif. Intell.*, vol. 26, no. 5–6, pp. 1575–1587, May 2013.

[141] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” *CoRR*, vol. abs/1312.5, 2013.

[142] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[143] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource Management with Deep Reinforcement Learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[144] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end Training of Deep Visuomotor Policies,” *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016.

[145] Z. Zhou, X. Li, and R. N. Zare, “Optimizing Chemical Reactions with Deep

Reinforcement Learning," *ACS Cent. Sci.*, vol. 3, no. 12, pp. 1337–1344, 2017.

[146] D. Silver, "Tutorial: Deep Reinforcement Learning," *Proc. Int. Jt. Conf. Neural Networks*, 2010.

[147] P. Dayan, "Unsupervised Learning," *MIT Encycl. Cogn. Sci.*, 2004.

[148] S. Kullback, *Information Theory and Statistics*. Dover Publications, 1997.

[149] M. I. Jordan, Ed., *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press, 1999.

[150] L. R. Squire, "Memory and Brain Systems: 1969{\textendash}2009," *J. Neurosci.*, vol. 29, no. 41, pp. 12711–12716, 2009.

[151] D. Derdikman and M.-B. Moser, "A Dual Role for Hippocampal Replay," *Neuron*, vol. 65, no. 5, pp. 582–584, 2010.

[152] M. F. Carr, S. P. Jadhav, and L. M. Frank, "Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval.," *Nat. Neurosci.*, vol. 14, no. 2, pp. 147–153, Feb. 2011.

[153] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, no. 3, pp. 293–321, May 1992.

[154] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *CoRR*, vol. abs/1511.0, 2015.

[155] T. Hester *et al.*, "Learning from Demonstrations for Real World Reinforcement Learning," *CoRR*, vol. abs/1704.0, 2017.

[156] M. Vecerik *et al.*, “Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards,” *CoRR*, vol. abs/1707.0, 2017.

[157] S. Thrun, “Lifelong Learning: A Case Study,” 1995.

[158] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on Knowledge and Data Engineering*. 2010.

[159] “Python,” *Python*, 2018. [Online]. Available: https://www.python.org/.

[160] S. Nanz and C. A. Furia, “A comparative study of programming languages in rosetta code,” in *Proceedings - International Conference on Software Engineering*, 2015.

[161] S. Balestrini-Robinson, D. F. Freeman, and D. C. Browne, “An object-oriented and executable SysML framework for rapid model development,” in *Procedia Computer Science*, 2015.

[162] MongoDB, “MongoDB,” *MongoDB*, 2018. [Online]. Available: https://www.mongodb.com/.

[163] MongoEngine, “MongoEngine,” 2018. [Online]. Available: http://mongoengine.org/.

[164] P. Cudré-Mauroux *et al.*, “NoSQL databases for RDF: An empirical evaluation,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013.

[165] Neomodel, “neomodel,” 2018. [Online]. Available:

https://neomodel.readthedocs.io/en/latest/.

[166] “neo4j,” *neo4j*, 2018. [Online]. Available: https://neo4j.com/.

[167] Mart\’\in~Abadi *et al.*, “{TensorFlow}: Large-Scale Machine Learning on Heterogeneous Systems.” 2015.

[168] Apache, “The Apache Software Foundation,” *Hadoop website*, 2014. .

[169] Django, “Django,” *The web framework for perfectionists with deadlines.*, 2016. .

[170] D. REST, “Django REST framework,” *Citirano 14.9.2014: http://www.django-rest-framework.org/#django-rest-framework*, 2014. .

[171] TensorFlow, “TensorBoard: Visualizing Learning,” *TensorFlow*, 2017. .

[172] D. Hess and M. Summerfield, “PyQt Whitepaper,” *Riverbank Comput. Ltd.*, 2013.

[173] INCOSE, “Systems Engineering Vision 2020,” *Systems Engineering Vision 2020*, 2007. .

[174] Nomagic.com, “MagicDraw,” *No Magic, Inc.* [Online]. Available: https://www.nomagic.com/products/magicdraw.

[175] S. Systems, “Enterprise Architect,” *EnterpriseArchitect*, 2018. [Online]. Available: https://sparxsystems.com/products/ea/.

[176] N. S. E. Division, “Final Report of the Model Based Engineering (MBE) Subcommittee,” 2011.

[177] Z. C. Fisher, D. Locascio, K. D. Cooksey, D. N. Mavris, and E. Spero, “ADAPt

Design: A Methodology for Enabling Modular Design for Mission Specific SUAS," in *Volume 2B: 42nd Design Automation Conference*, 2016.

[178] D. Locascio, C. Ramee, E. Schaus, K. D. Cooksey, E. Spero, and D. N. Mavris, "A Framework for Integrated Analysis, Design, and Rapid Prototyping of Small Unmanned Airplanes," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, American Institute of Aeronautics and Astronautics, 2016.

[179] J. Humann and E. Spero, "Modeling and simulation of multi-UAV, multi-operator surveillance systems," in *2018 Annual IEEE International Systems Conference (SysCon)*, 2018, pp. 1–8.

[180] C. Justin, A. Ramamurthy, N. Beals, E. Spero, and D. Mavris, "On-Demand Small UAS Architecture Selection And Rapid Manufacturing Using A Model-Based Systems Engineering Approach," in *31st Congress of the International Council for Aeronautical Sciences*, 2018.

[181] L. E. Hart, "Introduction To Model-Based System Engineering (MBSE) and SysML," 2015. [Online]. Available: www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf.

[182] J. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," in *INCOSE MBSE Focus Group*, 2007.

[183] K. A. Reilley, S. Edwards, R. Peak, and D. Mavris, "Methodologies for Modeling and Simulation in Model-Based Systems Engineering Tools," in *AIAA SPACE 2016*, American Institute of Aeronautics and Astronautics, 2016.

[184] A. A. Yassine, “An Introduction to Modeling and Analyzing Complex Product Development Processes Using the Design Structure Matrix (DSM) Method,” *Urbana*, 2004.

[185] T. R. Browning, “Applying the design structure matrix to system decomposition and integration problems: A review and new directions,” *IEEE Transactions on Engineering Management*. 2001.

[186] K. Forsberg and H. Mooz, “The relationship of systems engineering to the project cycle,” *EMJ - Eng. Manag. J.*, 1992.

[187] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.

[188] S. G. MacDonell, K. Min, and A. M. Connor, “Autonomous requirements specification processing using natural language processing,” *CoRR*, vol. abs/1407.6, 2014.

[189] A. G. Arellano, “Frameworks for Natural Language Processing of Textual Requirements,” 2018.

[190] M. De Marneffe and C. D. Manning, “Stanford typed dependencies manual,” *20090110 Httpnlp Stanford*, 2015.

[191] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc.,

2013, pp. 3111–3119.

[192] “nummod: Numeric Modified.” [Online]. Available: http://universaldependencies.org/en/dep/nummod.

[193] J. D. Mattingly, W. H. Heiser, and D. T. Pratt, *Aircraft Engine Design - Second Edition*. 2002.

[194] C. B. Browne *et al.*, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[195] A. G. Barto and S. Mahadevan, “Recent Advances in Hierarchical Reinforcement Learning,” *Discret. Event Dyn. Syst.*, vol. 13, no. 1–2, pp. 41–77, Jan. 2003.

[196] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 3675–3683.

[197] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artif. Intell.*, vol. 112, no. 1, pp. 181–211, 1999.

[198] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning,” *CoRR*, vol. abs/1509.0, 2015.

[199] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in *Proceedings of the 27th International Conference on International*

Conference on Machine Learning, 2010, pp. 807–814.

[200] D. P. Kingma and J. Ba, “Adam: {A} Method for Stochastic Optimization,” *CoRR*, vol. abs/1412.6, 2014.

[201] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *CoRR*, vol. abs/1509.0, 2015.

[202] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[203] D. C. Montgomery, *Design and Analysis of Experiments*. USA: John Wiley & Sons, Inc., 2006.

[204] S. PLM, “Siemens NX,” *Siemens*, 2018. .

[205] S. PLM, “Knowledge Fusion,” *NX Programming Tools*. [Online]. Available: https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_api#uid:index_fusion:id1395371:intro_int_ov.

[206] D. Systemes, “KnowledgeWare,” *CATIA*. [Online]. Available: http://catiadoc.free.fr/online/cfyugkwr_C2/cfyugkwr3002.htm.

[207] K. S. East, “CommunityCommands : Command Recommendations for Software Applications,” pp. 193–202.

[208] W. Li, J. Matejka, T. Grossman, and G. Fitzmaurice, “Deploying CommunityCommands :,” no. Twidale 2005, pp. 19–34, 2015.

[209] W. Li, J. Matejka, T. Grossman, and G. Fitzmaurice, “Deploying

CommunityCommands: a software command recommender system case study," *AI Mag.*, 2015.

[210] J. Matejka and W. Li, "CommunityCommands: command recommendations for software applications," in *Proceedings of the ACM conference on User interface software and technology*, 2009.

[211] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice, "CommunityCommands," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology - UIST '09*, 2009.

[212] W. Li, J. Matejka, T. Grossman, J. A. Konstan, and G. Fitzmaurice, "Design and evaluation of a command recommendation system for software applications," *ACM Trans. Comput. Interact.*, vol. 18, no. 2, pp. 1–35, 2011.

[213] Simran Wasu, "GDSTIME 50mm x 50mm x 20mm 12V Brushless Cooling Fan," *grabcad.com*, 2018. [Online]. Available: https://grabcad.com/library/gdstime-50mm-x-50mm-x-20mm-12v-brushless-cooling-fan-1.

[214] J. Pasha, "Turbofan Engine," *grabcad.com*, 2018. [Online]. Available: https://grabcad.com/library/turbofan-engine-5.

[215] S. PLM, "NX Open," *https://docs.plm.automation.siemens.com*. [Online]. Available: https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_api#uid:index_nxopen_prog_guide:id1142156:purpose.

[216] S. PLM, "SNAP," *https://docs.plm.automation.siemens.com*. [Online]. Available:

https://docs.plm.automation.siemens.com/data_services/resources/nx/10/nx_api/en_US/graphics/fileLibrary/nx/snap/SNAP_Getting_Started_V10.pdf.

[217] S. PLM, "GRIP," *https://docs.plm.automation.siemens.com*. [Online]. Available: https://docs.plm.automation.siemens.com/data_services/resources/nx/10/nx_api/en_US/custom/grip/index.html.

[218] S. PLM, "User Exits," *https://docs.plm.automation.siemens.com*. [Online]. Available: https://docs.plm.automation.siemens.com/data_services/resources/nx/11/nx_api/custom/en_US/grip/user_exits/uex_ov.html.

[219] S. PLM, "Menuscripts," *https://docs.plm.automation.siemens.com*. [Online]. Available: https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_api#uid:index_menuscript:introduction_intro_menuscript.

[220] D. J. Jackson and D. J. Jackson, "Boundary representation modelling with local tolerances," *SMA '95 Proc. Third Symp. Solid Model. Appl.*, pp. 247–254, 1995.

[221] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice (2Nd Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[222] G. Valiente, *Algorithms on Trees and Graphs*. Berlin, Heidelberg: Springer-Verlag, 2002.

[223] C. Ferreira, "Gene Expression Programming: a New Adaptive Algorithm for

Solving Problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001.

[224] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence (Studies in Computational Intelligence)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[225] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, 2014, p. II-1188--II-1196.

[226] L. J. P. van der Maaten and G. E. Hinton, "Visualizing High-Dimensional Data Using t-SNE," 2008.

[227] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *In Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, pp. 278–287.

[228] A. D. Laud, "Theory and Application of Reward Shaping in Reinforcement Learning," University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2004.

[229] M. Grześ, "Reward Shaping in Episodic Reinforcement Learning," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 2017, pp. 565–573.