

A Graphical Programming Interface for a XML-based Hidden Markov Model Compiler



Ahmad Usman Chughtai

St. Hugh's College

University of Oxford

A thesis submitted in partial fulfilment of
the requirements for the degree of

Master of Science in Computer Science

Oxford University Computing Laboratory

Trinity 2005

Abstract

Hidden Markov Models (HMMs) are simple yet powerful statistical models to describe sequential data. First introduced in the context of speech recognition, they are now widely used and are particularly successful in Bioinformatics. Applications include modelling of protein domains, gene finding, probabilistic sequence alignment and DNA binding site modelling.

Because of the size of typical biological data sets (e.g. the human genome consists of 2.85 billion base pairs), efficiency issues are crucial. On the other hand, the rate at which new insights and discoveries are published means that we'd like to rapidly test new probabilistic models, requiring a great deal of flexibility. These two requirements are often conflicting. Moreover, the algorithms themselves are known in detail and although tedious and error-prone, implementing them is largely an 'automatic' process.

This characteristic was exploited to solve the conflict between the requirements, by building a "HMM compiler", termed HMMoC. This Java program reads a high-level description of the hidden Markov model in XML format, making it very easy to change the underlying model. It produces highly efficient C++ code for the various probabilistic algorithms (Viterbi, Forward/Backward etc.). The resulting code is faster than most hand-coded implementations.

The remaining drawback of this approach is that it requires the user to learn a new (XML) language. The natural way to specify a HMM would be to draw a graph representing the Markov chain, specify transition and emission probabilities and have the computer convert the graph into XML. This would be a valuable research tool and would be even more valuable in teaching.

A Graphical Programming Environment for specifying and manipulating HMMs termed "G-HMMoC" has been developed to act as a graphical layer between the user and the XML specification. The process detailing the analysis, design and implementation of the application is discussed, in addition to its success in modelling HMMs and translating them into an equivalent XML specification recognised by HMMoC.

Acknowledgment

The author wishes to express his sincere gratitude to Prof. Peter Jeavons and Prof. Jotun Hein for their constant support and encouragement during the course of this project. A very special thanks to Dr. Gerton Lunter for spending much time in explaining the design of *HMMoC* and making it look “simple” and also to Dr. Rune Lyngso for suggesting many interesting design features for the application, reviewing the work and offering valuable feedback.

Table of Contents

1	Introduction.....	1
1.1	Background.....	1
1.2	Statement of the Problem.....	3
1.3	Objectives.....	3
1.4	Motivation.....	4
1.5	Report Structure.....	5
2	Hidden Markov Models.....	6
2.1	Markov Chains.....	6
2.2	Formal Definition of HMM.....	8
3	Requirements.....	9
3.1	Techniques.....	9
3.2	Approach.....	12
3.2.1	<i>Requirements Gathering</i>	13
3.2.2	<i>Design</i>	13
3.2.3	<i>Prototype Building</i>	13
3.2.4	<i>Customer Evaluation</i>	13
3.2.5	<i>Prototype Refinement</i>	13
3.2.6	<i>Product Engineering</i>	13
3.3	Functional Requirements.....	14
3.3.1	<i>HMM Settings</i>	14
3.3.2	<i>HMM Objects</i>	14
3.3.3	<i>HMM Object Properties</i>	15
3.3.4	<i>HMM File Input/Output</i>	15
3.4	Non-functional Requirements.....	15
4	Design.....	16
4.1	Methodology.....	16
4.2	Interface.....	17
4.2.1	<i>Menus</i>	17
4.2.2	<i>Toolbar</i>	18
4.2.3	<i>Chart</i>	18
4.2.4	<i>Properties and Settings</i>	18
4.3	HMM Composition.....	18
4.3.1	<i>Alphabet</i>	18
4.3.2	<i>Sequences</i>	18
4.3.3	<i>Algorithms</i>	19
4.3.4	<i>Emission Probability Distributions</i>	19
4.3.5	<i>State</i>	19
4.3.6	<i>Transition</i>	21
4.4	Saving/Loading to/from File.....	22
4.4.1	<i>Alphabet</i>	23
4.4.2	<i>Sequences</i>	23
4.4.3	<i>Algorithms</i>	23
4.4.4	<i>Emission Tables</i>	24
4.4.5	<i>State</i>	25
4.4.6	<i>Transition</i>	26

4.5	HMMoC Input Syntax and Semantics	27
4.5.1	<i>Definition of HMM</i>	27
4.5.2	<i>Specification of Algorithms</i>	31
4.6	Model-View-Controller Architecture	32
4.7	Objected Oriented Analysis and Design	33
5	Implementation	34
5.1	Java 2D	34
5.2	Interface	35
5.3	HMM Settings	36
5.3.1	<i>Alphabet</i>	36
5.3.2	<i>Sequences</i>	36
5.3.3	<i>Algorithms</i>	37
5.3.4	<i>Emission Tables</i>	37
5.4	HMM Properties	39
5.4.1	<i>State</i>	39
5.4.2	<i>Transition</i>	44
5.5	Observer/Observable Design Pattern	49
6	Evaluation	50
6.1	HMM Settings	51
6.1.1	<i>Alphabet</i>	51
6.1.2	<i>Sequences</i>	51
6.1.3	<i>Algorithms</i>	51
6.1.4	<i>Emission Tables</i>	51
6.2	HMM Drawing	52
6.3	HMM Object Properties	53
6.4	Save to File	53
6.5	Generate XML	53
7	Conclusion	57
8	Future Development	59
8.1	Chart Settings	59
8.2	Multiple Selected Objects Properties	59
8.3	Clipboard Support	59
8.4	Consistency Check	60
8.5	State Shapes	60
8.6	Repetitive Structures	60
8.7	One-Touch Compilation	60
8.8	Results	60
A	Use Case Analysis	61
A.1	Define Alphabet	61
A.2	Specify Sequences	62
A.3	Specify Algorithms	62
A.4	Define Probability Distribution	63
A.5	Select Probability Distribution	63
A.6	Delete Probability Distribution	64
A.7	Insert State	64
A.8	Select Single State	65
A.9	Select Multiple States	66
A.10	Change State Label	67
A.11	Change State Type	67
A.12	Change State Appearance	68

A.13	Assign Emission Probability Distribution	68
A.14	Change State Shape.....	69
A.15	Insert Transition	69
A.16	Move State	70
A.17	Delete State	70
A.18	Select Single Transition	71
A.19	Select Multiple Transitions	72
A.20	Delete Transition.....	73
A.21	Set Transition Probability	73
A.22	Move Transition.....	74
A.23	Change Transition Shape	75
A.24	Change Transition Appearance.....	75
A.25	Create new File	76
A.26	Save HMM to File	76
A.27	Open HMM from File.....	77
A.28	Generate XML File	77
B	UML Class Diagram	78
C	Casino HMM Saved File	79
	Bibliography	86

List of Figures

Figure 1.1 - Existing Operation of HMMoC	2
Figure 1.2 - Proposed Operation of HMMoC	3
Figure 2.1 - Markov Chain for DNA	7
Figure 3.1 - A HMM constructed by HMMER	10
Figure 3.2 - HMMpro Interface	11
Figure 3.3 - The Prototyping Model	12
Figure 4.1 - Iterative Design Methodology.....	16
Figure 4.2 – G-HMMoC Interface Design Layout	17
Figure 4.3 - Model-View-Controller Interaction	32
Figure 5.1 – G-HMMoC Interface Implementation.....	35
Figure 5.2 - Alphabet Settings Panel	36
Figure 5.3 - Sequences Settings Panel	36
Figure 5.4 - Algorithms Settings Panel.....	37
Figure 5.5 - Single Sequence Emission Probability Distribution Table	37
Figure 5.6 - Double Pair Emission Probability Distribution Table	38
Figure 5.7 - Operation Panel for Adding/Removing Tables.....	38
Figure 5.8 - Selection Panel for Probability Distributions.....	39
Figure 5.9 - A Selected State	39
Figure 5.10 - Multiple Selections with the Mouse.....	40
Figure 5.11 - State General Properties Panel	40
Figure 5.12 - State Position Properties Panel.....	41
Figure 5.13 - State Type Properties Panel	41
Figure 5.14 - State Shape Properties Panel	41
Figure 5.15 - State Appearance Properties Panel.....	42
Figure 5.16 - State Colour Chooser Dialog	42
Figure 5.17 - State with modified Appearance Properties.....	43
Figure 5.18 - State Emission Probability Properties Panel	43
Figure 5.19 - A Transition between two States.....	44
Figure 5.20 - Selected Quadratic and Cubic Curve Transitions	44
Figure 5.21 - Transition State Properties Panel	45
Figure 5.22 - Transition Position Properties Panels.....	45
Figure 5.23 - Transition Behaviour Properties Panel.....	46
Figure 5.24 - Transition Type Properties Panel	46
Figure 5.25 - State with Self Transition.....	46
Figure 5.26 - Transition Shape Properties Panel	47
Figure 5.27 - Line, Quadratic Curve and Cubic Curve Transitions.....	47
Figure 5.28 - Transition Appearance Properties Panel	48
Figure 5.29 - Transition Probability Properties Panel	48
Figure 5.30 - Observer/Observable Design Pattern for G-HMMoC.....	49
Figure 6.1 - Fair and Loaded Die.....	50
Figure 6.2 - Defining Alphabets for Casino.....	51
Figure 6.3 - Emission Probabilities for Casino	52
Figure 6.4 - HMM for Casino	52
Figure 6.5 - Position Properties for Fair and Loaded States	53

Chapter One

Introduction

Hidden Markov Models (HMMs) are simple yet powerful statistical models to describe sequential data. In particular, HMMs can model matches, mismatches, insertions and deletions of symbols. They have been deeply rooted in computational linguistics and speech recognition where the main problem is huge variations in phonemes that are spoken in a particular time frame. This problem is dealt with by taking a spoken word and trying to fit it to a specific model of possible words.

Although first introduced and implemented in the context of speech recognition, HMMs are now widely used and are particularly successful in Bioinformatics where the problem of sequence analysis is similar. For instance, given an amino acid sequence, one may want to determine the protein family to which it belongs. Hence, the amino acid sequence may be thought of as the speech signal in a given frame and the amino acids may be treated as the phonemes.

Because of the size of typical biological data sets (e.g. the human genome consists of 2.85 billion base pairs), efficiency issues are crucial. On the other hand, the rate at which new insights and discoveries are published means that we'd like to rapidly test new probabilistic models, requiring a great deal of flexibility. These two requirements are often conflicting. Moreover, the algorithms themselves are known in detail and although tedious and error-prone, implementing them is largely an 'automatic' process.

1.1 Background

Dr. Gerton Lunter at the Oxford Centre for Gene Function has attempted to solve the conflict between these requirements, by building a "HMM compiler", termed *HMMoC*. This Java program reads a high-level description of the HMM in XML format, making it very easy to change the underlying model. It produces highly efficient C++ code for the various probabilistic algorithms (Viterbi, Forward/Backward etc.). The resulting code is faster than most hand-coded implementations.

HMMoC translates a high-level description of an arbitrary HMM into C++ code for computation of likelihoods, posterior probabilities, path decoding and training. It was designed in order to deal with HMMs occurring in computational biology but it is completely general and not tied to any particular application.

A list of features supported by the compiler as documented in its manual are:

- Multiple output tapes (“pair HMMs”, “triple HMMs” etc.)
- Mealy and Moore machines (emissions are associated to states or transitions)
- Higher-order states (transitions/emissions may depend on previous emitted symbols)
- Forward, Backward, Viterbi and Baum-Welch algorithms
- State grouping for better memory efficiency
- Extended-exponent reals
- XML Macro facilities

In addition, the following list of features is intended to be included:

- Banding
- Partially unobserved output
- Position-dependent transition and emission probabilities
- Memory-efficient Forward/Backward algorithm
- Hirschberg algorithm; bushy Viterbi algorithm

The existing operation of HMMoC is represented in Figure 1.1. The program expects a structured representation of a HMM as input in XML format and produces highly efficient C++ code as output. This C++ is later compiled and executed to perform various tasks.

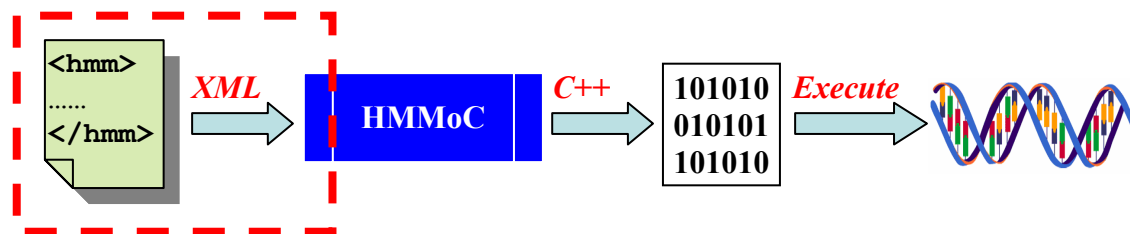


Figure 1.1 - Existing Operation of HMMoC

The scope of analysis of this project is represented by the dotted rectangle. The emphasis has been placed on the rules of specification of the HMM in XML format. The inner operation of the HMMoC and the generated C++ code are beyond the scope of this project and will not be discussed further.

1.2 Statement of the Problem

This technique of specifying HMMs in textual format is not without its drawbacks, the prime one being that it requires the user to learn a new (XML) language. In addition, the natural way to specify a HMM is to draw a graph representing the Markov chain, specify transition and emission probabilities in an application and have it the convert the graph into XML. It is believed that this will prove to be a valuable research tool and would be even more valuable in teaching.

1.3 Objectives

This project proposes to write such a Graphical Programming Interface, as a graphical layer between the user and the XML specification. It is expected that the graphical interface will be written in Java. The application whose scope of operation is represented by the dotted rectangle in Figure 1.2 has been termed *Graphical Hidden Markov Model Compiler* or *G-HMMoC* in short.

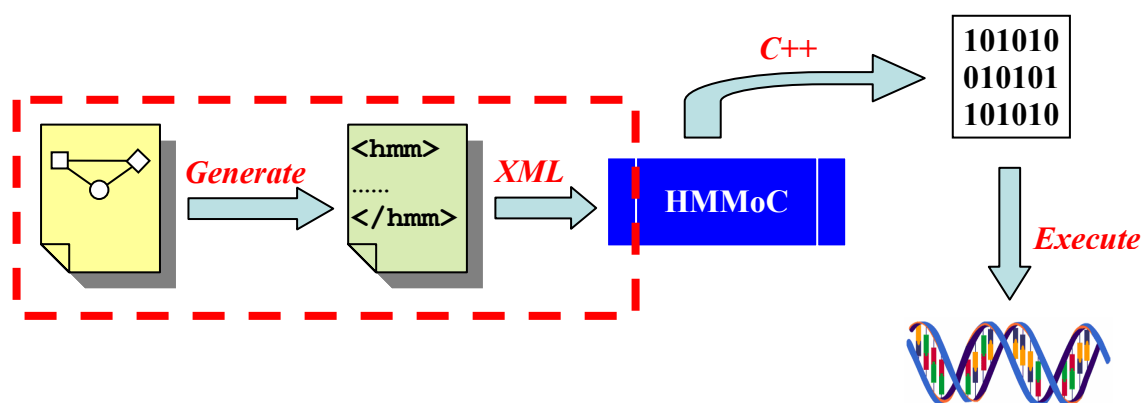


Figure 1.2 - Proposed Operation of HMMoC

In order to achieve the main objective a number of distinct tasks have been identified.

- Development of the Graphical Drawing Tool
- Development of the Properties/Manipulation Tool
- Provision for Saving/Loading HMM in Graphical Format to/from File
- Translation of HMM in Graphical Format to XML

1.4 Motivation

The motivation behind this project arises from the lack of popularity of HMMoC amongst users, because of its unnatural method of textual specification and the outlook of learning a complex set of specification rules.

In addition, there are very few graphical editors available that can be used to specify HMMs in a visually rich environment. This factor lends the potential, of a solution to this problem to be used also in other disciplines and particularly for research and teaching.

The project also relates to the author's present research interests, in the use of HMMs, both in computational linguistics and gene analysis. In addition, the main requirement of the graphical interface provides an opportunity to be artistic and creative in solving this open problem. There is a further scope of ideas to be explored, in terms of effectiveness and efficiency in XML output generation.

The project further links closely with material studied during the taught part of the course. The following modules undertaken during the MSc have directly influenced and played a pivotal role in the research and development of this project.

1. *Computational Linguistics*

This course although not directly related to the product domain lent an introduction to HMMs and their application in speech recognition.

2. *Requirements*

This course addressed the problem of determining requirements in a wide array of scenarios and taught traditional software engineering and more modern techniques including the use of UML diagrams and Use Cases. These techniques were directly applied during the requirements capture and design phases of this project.

3. *Object Oriented Programming I*

This course taught some advanced objected-oriented design and modelling techniques in Java including design patterns such as the Model-View-Controller Design Pattern. These techniques were applied during the development phase and proved to be extremely beneficial.

1.5 Report Structure

Chapter 2 presents a brief discussion of HMMs, their characteristics and properties and their usage in solving real problems.

Chapter 3 presents an analysis to determine the requirements of a system capable of modelling HMMs effectively.

Chapter 4 discusses the design details of the G-HMMoC in depth with a detailed analysis of the rules of specification of HMMoC in XML format.

Chapter 5 presents a detailed account of how the system was implemented to meet the design specification outlined in the previous chapter.

Chapter 6 presents a worked example of a HMM beginning from its graphical specification in G-HMMoC through to translation into XML.

Chapter 7 contains the final conclusions about the project, summarises the approach taken and highlights the main achievements.

Chapter 8 explores avenues for further development and suggests a number of additional features that may be added to G-HMMoC to enhance its functionality.

Chapter Two

Hidden Markov Models

This chapter presents a brief discussion of HMMs, their characteristics and properties and their usage in solving real problems. It is intended for the unfamiliar reader as a basic introduction to the concept of HMMs and also as a starting point for introspection of the requirements and design of a system capable of modelling them efficiently in a visual environment.

The emphasis has been placed on their use in analysing Biological sequences with respect to the context of this project. The material is based largely on the text from Durbin, Eddy, Krogh and Mitchison ^[6].

2.1 Markov Chains

A Markov process also referred to as the Markov chain is a probabilistic model of a sequence of random variables where the probability of a symbol, depends upon the previous symbol in the sequence. The range of the variables is called the *State Space*.

The property that the current state depends on a finite history of previous states is referred to as the *Markov Assumption* or the *Markov Property*. A Markov chain can thus be characterised as follows:

$$P(X_{n+1}/X_n)$$

where, X_n is the state of the process at instance n . This is referred to as the *transition probability* of the process.

The simplest example of a Markov chain that we encounter in our everyday lives is the traffic lights.

Markov chains exist in many forms, the simplest of which is the *first-order* Markov chain, in which the current state depends only on the previous state and not on any earlier states.

Graphically, a Markov chain can be represented as a collection of states with arrows between them. A Markov chain for DNA is shown in Figure 2.1.

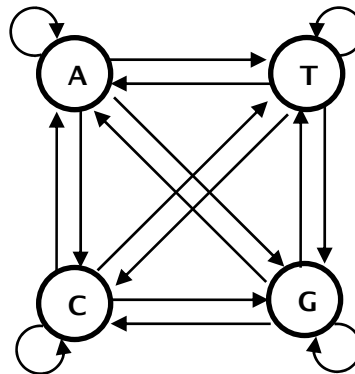


Figure 2.1 - Markov Chain for DNA

where, we see a state for each of the four letters A, C, G and T in the DNA *alphabet*. A transition probability is associated with each arrow in the arrow in the figure which determines the probability of a certain residue following another residue, or one state following another state. ^[6]

Hence, summing up on the above a Markov chain is simply a set of states S_i ($i = 0..n$) connected by transitions, with a transition from state S_i to state S_j having a probability P_{ij} .

Given, the Markov chain in Figure 2.1, the probability of a sequence, GCCT is calculated as:

$$P(GCCT) = P(T|C) P(C|C) P(C|G) P(G)$$

HMMs are an extension of Markov chains and the essential difference between them is that for a HMM there is not a one-to-one correspondence between the states and the symbols. It is no longer possible to tell what state the model was in when x_i was generated just by looking at x_i ^[6]. Each state has a distribution of symbols from amongst the alphabet associated with it. Hence, the state for a particular output symbol (also referred to as an *emission* symbol) is ambiguous or *hidden*.

2.2 Formal Definition of HMM

The following definition has been adapted from the notes of Singh ^[5]. A HMM may be defined by specifying the following:

$S = \text{the set of states} = \{s_1, s_2, \dots, s_n\}$

$A = \text{the output alphabet, also referred as the set of emission symbols} = \{a_1, a_2, \dots, a_m\}$

$\pi(i) = \text{probability of being in state } s_i \text{ at time } t = 0 \text{ (i.e., in initial states)}$

$T = \text{transition probabilities} = \{p_{ij}\}$,
where, p_{ij} is the probability associated with a transition from state s_i to state s_j

$O = \text{emission probabilities} = \{e_j(k)\}$
where, $e_j(k)$ is the probability of emitting alphabet a_j in State s_k .

The probability of being in the initial states may be abstracted by defining a *Start* state which is *silent* (does not output any symbols) and assigning a probability value p_{0j} .

Chapter Three

Requirements

After providing a flavour of the usage and study of HMMs in the previous chapter, this chapter sets out in determining the requirements of a system capable of modelling them effectively.

The techniques used to carry out the requirements elicitation process are discussed and the approaches undertaken to refine and perfect them. Finally, a use case analysis of the process is undertaken, concluding with the final set of requirements for the system.

3.1 Techniques

The foremost technique used in the requirements capture for the system was an informal open-ended interview with the end-users of the system under development and the developers of the existing HMMoC. Ideas were explored by drawing graphical representations of HMMs on a whiteboard using open verbal protocols to identify the desirable features and to filter for the undesirable ones. This proved extremely successful in providing an initial flavour of the problem domain and the expectations of the users.

The second most obvious method to be employed in this respect was introspection. This involves imagining the requirements of the system from the user's viewpoint. The research on the use of HMMs as discussed in Chapter 2, proved particularly useful in this respect of determining both the graphical and internal structural requirements of the HMM.

Finally, a review of the functionality of existing drawing packages in particular HMM editors was carried out. The objective in this case was to design the object drawing behaviour to be consistent with the tools that the users would already be familiar with. An obvious choice for this purpose was Microsoft PowerPoint which is used by professionals of almost all disciplines in preparing presentations. The functionality associated with the insertion, deletion and selection of single or multiple objects and positioning them on the desired location on the chart, was therefore adopted from it.

There have not been many editors that have been specifically designed for the study and specification of HMMs visually. One particular editor that was particularly reviewed for its eye-catching graphical representation capabilities is *HMMER*.

HMMER is an implementation of profile HMM methods for sensitive database searches using multiple sequence alignments as queries. HMMER takes a multiple sequence alignment as input. It can then build a HMM, which can be used as a query into a sequence database to find (and/or align) additional homologues of the sequence family. ^[1]

A HMM for sequence alignment purposes as constructed by HMMER is shown in Figure 3.1.

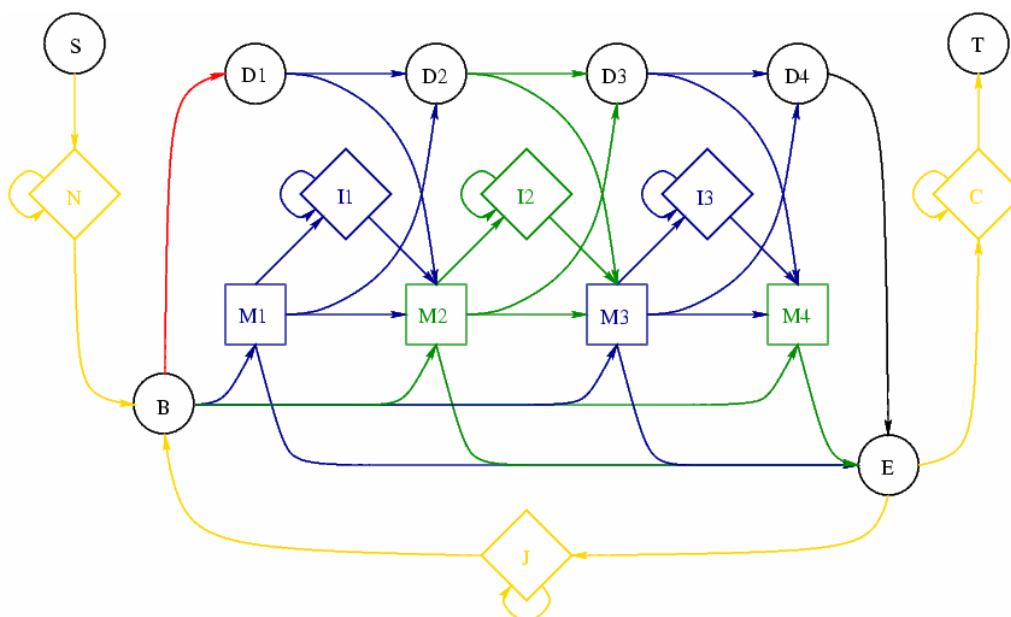


Figure 3.1 - A HMM constructed by HMMER

In the diagram, the circles stand for deletes, diamonds for insertions and squares for matches. The colouring represents some of the properties related to the internal representation of the program. This demonstrates a useful way of grouping similar states by assigning them the shapes or colours.

Another program that deserves a particular mention in this category is *HMMpro* shown in Figure 3.2. Developed by Net-ID ^[7], it is used across the bioinformatics, biotechnology and pharmaceutical industries as well as academic institutions. It is a commercial application but may be used under a free single-user academic licence.

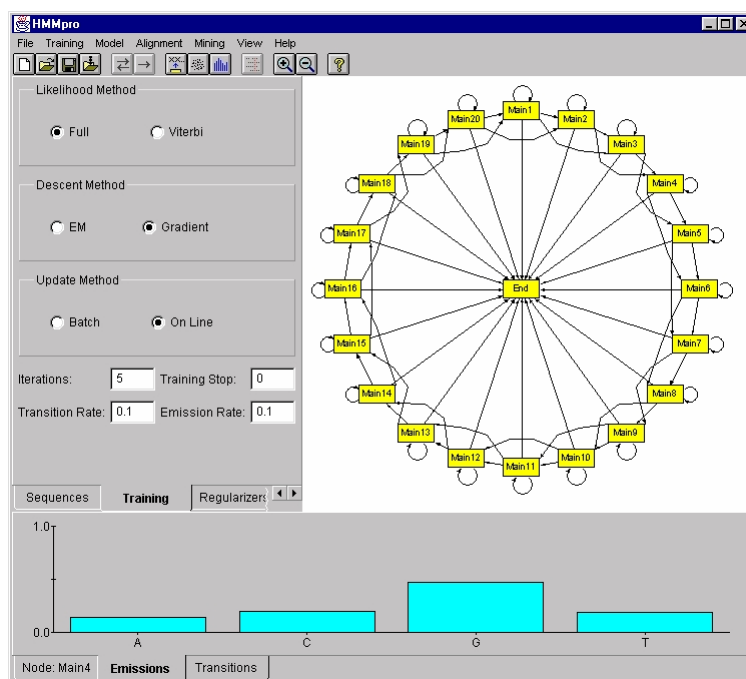


Figure 3.2 - HMMpro Interface

HMMpro is a general purpose HMM simulator for biological sequence analysis. It uses machine learning techniques to automatically build statistical models of protein and DNA sequences. These models can be used in a number of computational molecular biology tasks including: multiple alignments, data base searches, protein classification, gene finding, and the discovery of new patterns

The most impressive feature about HMMpro is its interface which lays out the drawing, manipulation and display tools in an extremely convenient manner with most functions only a single mouse-click away from the user.

3.2 Approach

Given the above scenario, the prototyping model of software was deemed to be most suitable for this project. The distinctive characteristic of this model is that it is most useful where the requirements are not entirely clear at the beginning and uses a prototype to refine and perfect them.

This character of the prototyping approach is also confirmed by [2] as follows,

A prototype information system is a working model of an information system built to learn about its true requirements. Firms build prototypes to solve a knowledge problem of not knowing exact what the system should do to address an important issue. In this situation, the user needs some way to get a feeling for how the system should operate. Accordingly, a prototype is built quickly to help the user understand the problem and determine how an information system might help in solving it. ^[2]

There are different prototyping models proposed by researchers over the years. The prototyping approach used during the course of this project is represented in Figure 3.3. ^[3]

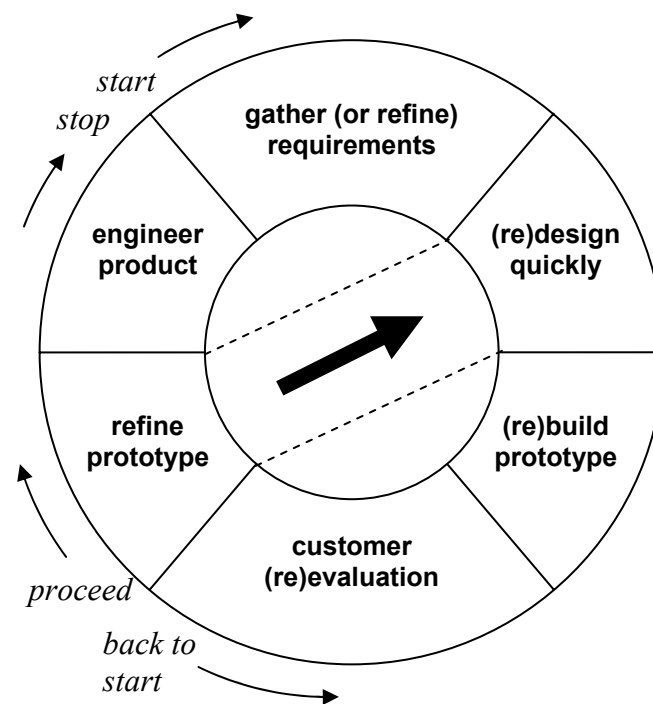


Figure 3.3 - The Prototyping Model

The process begins from the requirements gathering stage and iterates through various stages to deliver a prototype that will closely resemble the final system. A brief description of the process during each stage is discussed next.

3.2.1 Requirements Gathering

This stage marked the start of the entire process and its purpose was to obtain a thorough understanding of the domain. This was achieved through the various requirements elicitation techniques discussed under Section 3.1 .

The emphasis during this stage was *what* needs to be achieved as opposed to *how* it is to be done. The concern was with both what there is already i.e. the *existing* operation of HMMoC and what there should be i.e. the *desired* operation.

3.2.2 Design

This stage is concerned with the design of the prototype based on the requirements gathered in the previous stage. The application interface and interaction was granted the highest priority.

3.2.3 Prototype Building

During this stage a *mock* application was built based on the requirements analysis and design performed in the previous stages. The purpose of this application was for evaluation purposes only.

3.2.4 Customer Evaluation

The prototype constructed during the previous stage was evaluated and the comments received were fed back into the previous stages. This is an important stage as it requires one to assess the existing state of the prototype and decide whether to proceed or go back and improve further.

3.2.5 Prototype Refinement

This stage was and can only be reached once the prototype and the gathering of the requirements had reached a satisfactory standard. The purpose of this stage was to incorporate the final comments received from the previous stage into arriving at the final requirements of the system.

3.2.6 Product Engineering

This stage is concerned with the final design, implementation and testing of the system. The prototype produced as a result was used directly in the design and implementation of the final product.

3.3 Functional Requirements

This section lists the finalised functional requirements for the project. A detailed use case analysis for the process requirements is given in Appendix A.

3.3.1 HMM Settings

1. The system shall allow the user to specify an *Alphabet* for the HMM.
2. The system shall allow the user to set the number of *sequences* of emission symbols output by the HMM.
3. The system shall allow the user to select the *Algorithms* applicable to the HMM.
4. The system shall allow the user to *Define Emission Probability Distribution Tables* which may be assigned to any State.
5. The system shall allow the user to *Select* an existing *Emission Probability Distribution Table* for viewing.
6. The system shall allow the user to *Edit* an existing *Emission Probability Distribution Table*.
7. The system shall allow the user to *Delete* an existing *Emission Probability Distribution Table*.

3.3.2 HMM Objects

1. The system shall allow the user to *Insert* a *State* at any position in the Chart.
2. The system shall allow the user to *Select* any *State* that has been defined in the Chart.
3. The system shall allow the user to *Select Multiple States* that have been defined in the Chart.
4. The system shall allow the user to *Insert* a *Transition* between two States.
5. The system shall allow the user to *Move* a *State* to any position in the Chart.
6. The system shall allow the user to *Delete* an existing *State* from the Chart.
7. The system shall allow the user to *Select* any *Transition* that has been defined in the Chart.
8. The system shall allow the user to *Select Multiple Transitions* that have been defined in the Chart.
9. The system shall allow the user to *Delete* an existing *Transition* from the Chart.
10. The system shall allow the user to *Move* a *Transition* to any position along the boundary of its Source or Destination States.

3.3.3 *HMM Object Properties*

1. The system shall allow the user to assign or change the *Label* of a *State*.
2. The system shall allow the user to change the *Type* of a *State*.
3. The system shall allow the user to change the *Appearance* of a *State*.
4. The system shall allow the user to *Assign* an Emission Probability Distribution to a *State*.
5. The system shall allow the user to change the *Shape* of a *State*.
6. The system shall allow the user to *Set* the *Probability* of a *Transition*.
7. The system shall allow the user to change the *Shape* of a *Transition*.
8. The system shall allow the user to change the *Appearance* of a *Transition*.

3.3.4 *HMM File Input/Output*

1. The system shall allow the user to *Create a new File* for editing.
2. The system shall allow the user to *Save the File*.
3. The system shall allow the user to *Open* a saved *File* for editing.
4. The system shall allow the user to *Generate XML* output for HMMoC for the HMM that has been defined in the Chart.

3.4 **Non-functional Requirements**

1. The system shall be programmed using the Java Programming Language.
2. The system shall be easy to use and have a simple layout of components on the screen.
3. The properties information of the selected object in the HMM must be displayed at all times in the Editor.

Chapter Four

Design

After having outlined the requirements of the system under development, in the previous chapter, we proceed to present a design for G-HMMoC.

This chapter begins by discussing the formal methodologies that were adopted to achieve a concrete model for implementation, proceeding onto discussing the various objects and components and their corresponding attributes. A detailed analysis of the syntactic and semantic rules of specification governing the input to HMMoC is covered in detail. Finally, a brief discussion is carried out on the use of the Model-View-Controller Architecture and design issues surrounding the development of the system.

4.1 Methodology

Since the information gathered from the requirements capture stage was not enough to offer a comprehensive knowledge of the problem domain, an *iterative design* approach was adopted to deliver a successful design of the system.

Iterative design is a design methodology based on a cyclic process of prototyping, testing, analyzing, and refining the work in progress. This is similar to the prototyping model used to determine and perfect the requirements, but the emphasis here is on achieving the right design for the system without compromising requirements. The process is modelled in Figure 4.1.

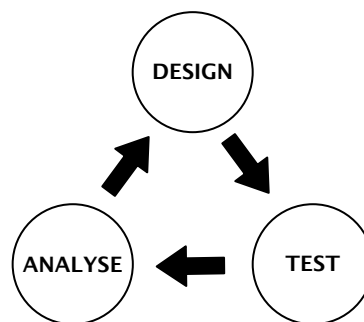


Figure 4.1 - Iterative Design Methodology

4.2 Interface

In order, for any software application to be exploited to its full potential, it is essential for the interface to be easy to learn, simple to use and straight forward to understand. Hence, the development of the interface requires a great deal of concern in software development and this case was no exception.

Since the aim of the project is to replace the textual representation of a HMM with a graphical one, it was essential that the interface must be elaborate enough to support all the textual features without adding additional complexity.

The primary concern when designing the interface was about the end-users who in this case are mostly Statisticians and Biologists with a non-computing background. It was therefore necessary to provide a simple layout using standard widgets with the majority of the functions accessible via a single mouse-click.

Figure 4.2 displays the overall layout of the interface of G-HMMoC and its division into various sections.

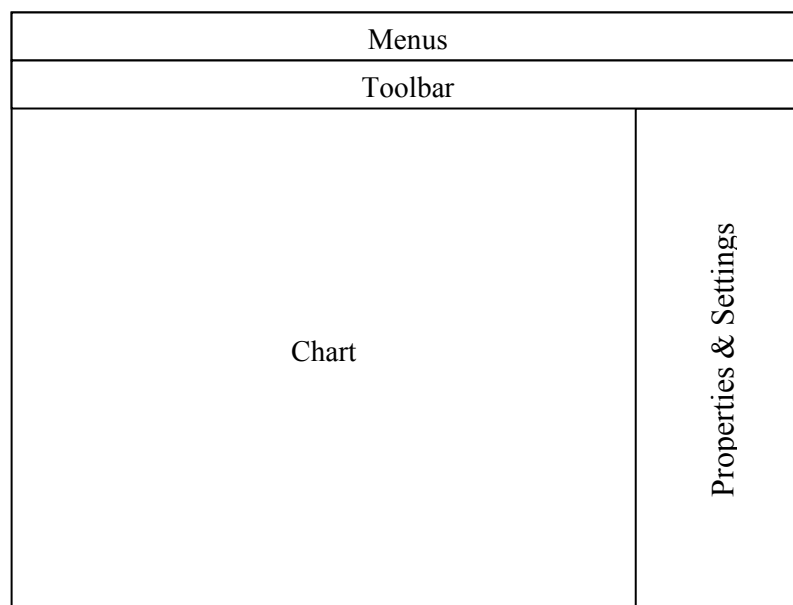


Figure 4.2 – G-HMMoC Interface Design Layout

A description of each of the sections above now follows.

4.2.1 Menus

The *Menus* provide a space-saving way to allow the user to choose one of several options. It provides access to all functions available within the program.

4.2.2 *Toolbar*

The *Toolbar* provides access to commonly used actions and functions. It contains a series of selectable buttons all grouped together and laid out horizontally.

4.2.3 *Chart*

The *Chart* is the main area of the application and provides the functionality of an editor. It is intended as a drawing and placement area for various objects that constitute a HMM.

4.2.4 *Properties and Settings*

The Properties and Settings area is split into two distinct sections accessed by means of tabs, which allows the user to switch between groups of components.

The *Settings* tab holds the components associated with the functionality and behaviour of the entire HMM as a whole.

The *Properties* tab is the default tab and contains the properties of the selected object on the Chart. This provides a means for the user to view all the information related to an object in the main window, without cluttering the Chart area.

4.3 HMM Composition

Section 2.1 provided a formal definition of a HMM and the various objects it comprises. This section discusses the composition of the HMM in detail and the relevant attributes of each object included as part of its design.

4.3.1 *Alphabet*

The alphabet of a HMM is the total set of unique symbols that may be observed in its output. These must be non-whitespace characters and must be specified before an emission probability distribution may be defined.

4.3.2 *Sequences*

This property relates to the number of output tapes that a state may emit at any given instance. These must be set at the beginning before specification and cannot be changed once an emission probability distribution has been defined. The minimum and default value for this is set at '1' with an upper limit of '3', as allowing higher values will not be computationally feasible.

4.3.3 Algorithms

The algorithms that are intended to be applicable to a HMM must be defined. Each algorithm is chosen by the user to be either enabled or disabled from amongst the following list each with their unique attributes:

- **Forward:** The *Forward* algorithm. The following parameters are defined:
 - Enabled/Disabled
 - Output Table Required / not Required
 - BaumWelch Applicable / not Applicable
- **Backward:** The *Backward* algorithm. The following parameters are defined:
 - Enabled/Disabled
 - Output Table Required / not Required
 - BaumWelch Applicable / not Applicable
- **Forward:** The *Viterbi* algorithm selected to be either enabled or disabled.
- **Sample:** The *Sample* algorithm selected to be either enabled or disabled.

4.3.4 Emission Probability Distributions

Once the alphabet and the number of output tapes have been finalised, the emission probability distribution tables may be added. Each table contains the set of symbols defined in the alphabet along with its corresponding probability set by the user. Leaving the probability field blank indicates that the particular symbol is not emitted in the particular distribution.

Further, each symbol may be paired with every other symbol depending on the numbers of output tapes that have been set. This is carried out automatically by the system. It is possible to remove a distribution, provided it is not assigned to any state. The probability values can be accessed and modified at any stage.

4.3.5 State

A HMM may consist of any number of States each with their own unique properties of which one must be a *Start* State and one an *End* State. The attributes of a State that are required to be set to construct a HMM is a unique *Label* for identification purposes and a probability distribution containing the corresponding probabilities for each symbol that it may emit. Hence, each State has the following attributes which are essential to its operation in the HMM:

- **Label:** A *String* value consisting of up to a maximum of 10 characters.
- **Type:** An *Integer* value identifying the role of the State in the HMM. This must be one of the following:
 - 0 = Other (*Any State in the HMM except Start and End*)
 - 1 = Start (*Start State*)
 - 2 = End (*End State*)
- **Probability:** An *Emission Probability Distribution* object that contains the symbols and their corresponding probabilities that may be emitted by this State.

In addition, the representation of the States in graphical form entails some additional attributes for each State for presentation purposes. These are:

- **Position:** A pair of *Integer* values corresponding to both the X and Y coordinates representing the point at the center of the State on the Chart with lower and upper limits constrained by its dimensions. This may be used to position the States neatly and align them horizontally or vertically. The position may also be changed by clicking on the State with the left-mouse button and dragging the mouse to the desired location on the Chart.
- **Shape:** The shape of the State may be calculated by checking the instance of the class that implements the State interface. This may be one of the following:
 - Circle (*CircleState*)
 - Diamond (*DiamondState*)
 - Square (*SquareState*)
- **Size:** An *Integer* value corresponding to the size of the graphical representation of the State on the Chart with allowable values between 1 and 30. The higher the value the larger the display on the Chart.
- **Line Thickness:** An *Integer* value representing the thickness of the outline of the graphical representation of the State on the Chart with allowable values between 0 and 10. The higher the value, the thicker the outline with a value of 0 displaying a dotted line.
- **Line Colour:** A *Color* object representing the colour of the outline of the graphical representation of the State on the Chart.
- **Fill Colour:** A *Color* object representing the colour of the background of the graphical representation of the State on the Chart.
- **Font Colour:** A *Color* object representing the colour of the text of the label of the State.

- **Font:** A *Font* object representing the font of the label of the State. The font may be chosen from a list of all fonts available on the System.
- **Font Size:** An *Integer* value representing the size of the font of the label of the State with allowable values between 10 and 72.
- **Bold/Italic:** A set of *Boolean* values that sets the font to Bold or Italic or both.

4.3.6 Transition

Each State may be linked to other States by means of a Transition. Thus each Transition in a HMM must be defined with respect to the States it connects. A Transition may also exist from a State to itself, referred to as a *Self Transition*. The direction of a transition is indicated by an arrow at the Destination end pointing towards it.

Since a Transition object is dependent for its existence on a set of Source and Destination States, deletion of either the Source or Destination States will automatically result in the Transition to be deleted as well. The attributes of a Transition that are essential to achieve the complete structure of a HMM are:

- **Source:** A *State* object referencing the Source State that this Transition is linked from.
- **Destination:** A *State* object referencing the Destination State that this Transition is linked to.
- **Probability:** A *Double* value corresponding to the size of the probability associated with the transition with allowable values between 0 and 1.

Further, as in the case with States, the representation of the Transitions in graphical form also entails some additional attributes for each Transition for presentation purposes. These are:

- **Position:** A pair of *Integer* values corresponding to both the X and Y coordinates of the end points of the Transition on the Chart with lower and upper limits constrained by the dimensions of the Source and Destination States respectively. For Quad Curve Transitions, an additional pair of coordinates is defined representing the single control point and an additional one for Cubic Curve Transitions. The position may also be changed by clicking on the end or control points with the left-mouse button and dragging the mouse to the desired location on the Chart. The end points are constrained by the System to the edge of the State.
- **Behaviour:** The behaviour controls adjust the response of the Transitions when their Source or Destination States are moved from their original position. All Transitions are locked to the centre point of their respective states

by default. This may however be changed by resetting the lock and positioning the end points along the edge of the State.

- **Type:** The type of the Transition is intended to be for display purposes only and may be one of the following:
 - Self (*Source and Destination States are the same*)
 - Other (*Source and Destination States are different*)
- **Shape:** The shape of the Transition is calculated by checking the instance of the class that implements the Transition interface. This may be one of the following:
 - Quadratic Curve (*QuadCurveTransition*)
 - Line (*LineTransition*)
 - Cubic Curve (*CubicCurveTransition*)
- **Line Thickness:** An *Integer* value representing the thickness of the outline of the graphical representation of the Transition on the Chart with allowable values between 0 and 3. The higher the value, the thicker is the outline with a value of 0 displaying a dotted line.
- **Line Colour:** A *Color* object representing the colour of the outline of the graphical representation of the Transition on the Chart.

4.4 Saving/Loading to/from File

Just as it was possible for HMM specification in the original XML format to be saved onto permanent storage for later retrieval, it was imperative that the same functionality be carried forward for a HMM to be viewed, modified and new XML output generated at a later date.

Since, the saving and loading of the file are inverse operations, it is important that all relevant attributes of HMM objects as defined in Section 4.3 must be saved to file in a manner that the exact same graphical representation be achieved upon loading.

In order to achieve this, a simple text file format, with an extension “.gmm” has been chosen. All relevant information is written to file encapsulated inside XML format tags for ease of parsing during retrieval.

Another possible solution that was considered at this stage was to use the existing XML definitions of HMMoC to also contain information about the graphical representation, thereby resulting in a single file which acts as input to both HMMoC and G-HMMoC. This approach was not adopted, as incorporating the visual elements along with the structural elements inside the XML input to HMMoC complicates the design and suffers from a greater possibility of error in both the syntax and semantics of the HMM definition.

The output produced for each component of the HMM and its representation in a saved file is now illustrated by consider various examples. The entire contents of the file are enclosed within the `<gmm>` `</gmm>` tags. The text shown in blue varies with the settings chosen by the user.

4.4.1 *Alphabet*

The file begins with storing the alphabet defined for the HMM. This is simply an array of characters on a single line enclosed with the `<alphabet>` `</alphabet>` tags. The following output is produced for an alphabet of nucleotides present in DNA.

```
<alphabets>  
  ACGT  
</alphabets>
```

4.4.2 *Sequences*

The number of output tapes emitted by the HMM are stored next. This is simply an integer enclosed with the `<sequences>` `</sequences>` tags as shown below.

```
<sequences>  
  1  
</sequences>
```

4.4.3 *Algorithms*

The algorithms and their properties applicable to the HMM are stored next. As each algorithm has a different set of properties they are stored inside their unique tags, which are all enclosed with the `<algorithms>` `</algorithms>` tags. The following output is produced for this case.

```
<algorithms>  
  <forward>  
    true  
    true  
    false  
  </forward>  
  <backward>  
    true  
    true  
    true  
  </backward>  
  <viterbi>  
    true  
  </viterbi>  
  <sample>  
    false  
  </sample>  
</algorithms>
```

The value for the *Viterbi* and *Sample* algorithms simply refer to whether the algorithm is applicable to the HMM or not. The *Forward* and *Backward* algorithm parameters are stored in the following order:

1. Algorithm Applicable
2. Output Table Generated
3. BaumWelch Enabled

4.4.4 *Emission Tables*

The emission probability distributions are represented in the program as tables. In the stored file, only the probability values of each alphabet or alphabet pairs (if applicable) are stored. This is because the alphabet order is static and can be regenerated from the alphabets and pairs information saved earlier. The name that has been assigned to the distribution is also stored.

The following code demonstrates the output for a table with four alphabet characters and a single pair. Any additional tables are defined in a similar manner and all of these grouped within the `<tables>` `</tables>` tags.

```
<table>
  PROB01
  0.1
  0.3
  0.4
  0.2
</table>
```

4.4.5 State

After storing the overall settings of the HMM, the complete list of states are saved enclosed within the `<states>` `</states>` tags. The output produced for saving a non-Start/End State with label, *State01* of shape circle position at X and Y coordinates of 100 and 200 on the chart with default appearance settings is given below and a probability distribution table *Prob01*.

```
<state>
  <label>
    STATE01
  </label>
  <type>
    0
  </type>
  <prob>
    PROB01
  </prob>
  <position>
    100
    200
  </position>
  <shape>
    0
  </shape>
  <appearance>
    1
    1
    -16777216
    -1
    -16777216
    Dialog
    10
    true
    false
  </appearance>
</state>
```

The colour attributes are stored as RGB values, so that the original *Color* object may be created on loading.

The *Appearance* settings are stored in the following order:

1. Size
2. Line Thickness
3. Line Colour
4. Fill Colour
5. Font Colour
6. Font Name
7. Font Size
8. Bold
9. Italic

4.4.6 Transition

The transitions between states are stored next enclosed within the `<transitions>` `</transitions>` tags. The output produced for a Cubic Curve Transition with `State01` and `State02` as source states on the chart with default behaviour and appearance settings and a probability of `0.7` is given below. Optional values that may not be applicable to all transitions are shown in red.

```
<transition>
  <state>
    STATE01
    STATE02
  </state>
  <prob>
    0.7
  </prob>
  <position>
    196
    193
    458
    322
    215
    307
    539
    239
  </position>
  <behaviour>
    true
    true
  </behaviour>
  <shape>
    0
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
```

The *States* are stored in the following order:

1. Source
2. Destination

The *Position* settings are stored in the following order:

1. Start X
2. Start Y
3. End X
4. End Y
5. Control Point One X (*if applicable*)
6. Control Point One Y (*if applicable*)
7. Control Point Two X (*if applicable*)
8. Control Point Two Y (*if applicable*)

The *Appearance* settings are stored in the following order:

1. Line Thickness
2. Line Colour

4.5 HMMoC Input Syntax and Semantics

The input to HMMoC is logically split into two parts: definition of the HMM and specification of the algorithms. The entire code is encapsulated inside the `<hml>` element with the *debug* option set to *true*. A static header listing the version and author is added to the start of each file as shown below:

```
<?xml version = "1.0"?>
<author> G-HMMoC </author>
```

There are numerous rules of specification but from amongst these, a small subset has been chosen that are sufficient to produce the desired output. The additional functionality that has been ignored was mainly associated with the use of macros and advanced features to make the textual representation easier and efficient to code. The use of the graphical interface renders these features redundant.

The rules governing the specification of a HMM in XML format that parse successfully are explained in the subsequent sections. For ease of understanding the code has been highlighted in different colours to identify its use. The static portion of the code that remains the same irrespective of the HMM being translated is shown in black. Variable parameters that are automatically generated by the system or input by the user as part of the graphical representation are shown in blue. Lastly, additional elements that may be included depending on some settings are given in red.

4.5.1 Definition of HMM

4.5.1.1 Alphabet

The *alphabet* element defines an alphabet of allowed emission symbols. All non-whitespace characters appearing as text are taken to be allowed symbols. The following output is produced for an alphabet for nucleotides.

```
<alphabet id = "alphabets">
  ACGT
</alphabet>
```

4.5.1.2 Output

The *output* element defines an emission tape referencing the alphabet defined previously and defines two identifiers of type *sequence* and *length* respectively. These specify the variables holding the length and content of the sequence, as an *integer* variable and a *character* array. The entire section of this code is static.

```
<output id = "sequenceX">
  <alphabet idref = "alphabets"/>
  <identifier type = "sequence" value = "aSeqX"/>
  <identifier type = "length" value = "iLenX"/>
  <code type = "parameter" value = "char* aSeqX"/>
  <code type = "parameter" value = "int iLenX"/>
</output>
```

The above code is repeated with unique labels and values for every additional emission tape depending on the number of sequences.

```
<output id = "sequenceY">
  <alphabet idref = "alphabets"/>
  <identifier type = "sequence" value = "aSeqY"/>
  <identifier type = "length" value = "iLenY"/>
  <code type = "parameter" value = "char* aSeqY"/>
  <code type = "parameter" value = "int iLenY"/>
</output>
```

4.5.1.3 Outputs

The *outputs* element contains reference to all output elements defined previously thereby listing all emission tapes for the HMM. The entire section of this code is static with additional sequences defined for each sequence.

```
<outputs id = "hmmoutputs">
  <output idref = "sequenceX"/>
  <output idref = "sequenceY"/>
</outputs>
```

4.5.1.4 Block

The *block* element lists the States contained in the HMM. Although it is permissible to group all the states in a single block, the Start and the End States are separated into their separate blocks as it results in a more efficient code to be produced by HMMoC. The following output demonstrates the code for a HMM with two States in addition to the standard Start and End States.

```
<block id = "block1">
  <state id = "START"/>
</block>

<block id = "block2">
  <state id = "STATE01" emission = "e1"/>
  <state id = "STATE02" emission = "xxx"/>
</block>

<block id = "block3">
  <state id = "END"/>
</block>
```

The *state id* is simply the label assigned to a given State. The *emission* attribute refers to the name of the emission probability distribution table associated with this state.

4.5.1.5 Graph

The *outputs* element contains reference to all block elements defined previously thereby grouping all states of the HMM. Since, the set of states are always split into the same three blocks, the entire content of this code is static.

```
<graph>
  <block idref = "block1"/>
  <block idref = "block2"/>
  <block idref = "block3"/>
</graph>
```

4.5.1.6 Transitions

The *transitions* element contains a list of all the transitions between the states along with their corresponding probabilities. Each probability is assigned a unique label automatically referencing a separate element containing the value. Continuing on the set of states used in the previous examples, the following piece of code demonstrates a graph with a single set of transitions linking each state.

```
<transitions>
  <transition from = "START" to = "STATE01" probability = "t1"/>
  <transition from = "STATE01" to = "STATE02" probability = "t2"/>
  <transition from = "STATE02" to = "END" probability = "t3"/>
</transitions>
```

4.5.1.7 Probability

The *probability* element simply lists the probability value next to the label assigned to each transition in the previous step. With the transition labels generated in the previous example, the code produced in this instance is given below.

```
<probability id = "t1">
  <code type = "expression"> X.X </code>
</probability>

<probability id = "t2">
  <code type = "expression"> Y.Y </code>
</probability>

<probability id = "t3">
  <code type = "expression"> Z.Z </code>
</probability>
```

4.5.1.8 Emission

The emission element specifies a particular emission on a set of output tapes. Each symbol or pairs of symbol lists the probability assigned to it. The quote tag of XML `<![CDATA["and"]]>` is used to insert code exactly as used by the compiler.

Pair HMMs reference multiple streams to calculate the final probability. The following code is produced for an alphabet consisting of nucleotides with the provision for multiple output tapes to be referenced (shown in red).

```
<emission id = "empty">
  <probability>
    <code type = "expression"> 1.0 </code>
  </probability>
</emission>

<emission id = "e1">
  <output idref="sequenceX"/>
  <output idref="sequenceY"/>
  <probability>
    <code type="statement">
      <identifier output = "sequenceX" value = "iEmissionX"/>
      <identifier output = "sequenceY" value = "iEmissionY"/>
      <identifier type = "result" value = "iProb"/>
      <![CDATA[
        switch(iEmissionxxx) {
          case 'A': iProb = 0.1; break;
          case 'C': iProb = 0.2; break;
          case 'G': iProb = 0.3; break;
          case 'T': iProb = 0.4; break;
          default: cerr << "Emission Table Error" << endl;
        }
      ]]>
    </code>
  </probability>
</emission>
```

4.5.2 Specification of Algorithms

4.5.2.1 Algorithms

This section specifies the algorithms to build for a particular HMM. The choice and settings of the algorithms is specified by the user, hence the entire section depends on whether the user has chosen the respective algorithm.

For the *forward* and *backward* algorithms, it is possible to specify whether the dynamic programming table should be retained or not. Retaining the table requires additional memory by the program generated by HMMoC, therefore the user may choose to turn it off at times. The table is at other times need for sampling and calculation of posterior likelihoods.

The *baumWelch* parameter may be selected for the forward or backward algorithms but not both at the same time.

```
<forward realtype = "bfloat" outputTable = "xxx" baumwelch = "xxx"
name = "Forward" id = "fw">
  <hmm idref = "xxx"/>
</forward>
```

```
<backward realtype = "bfloat" outputTable = "xxx" baumwelch = "xxx"
name = "Backward" id = "bw">
  <hmm idref = "xxx"/>
</backward>
```

```
<viterbi realtype = "bfloat" name = "viterbi" id = "vit">
  <hmm idref = "xxx"/>
</viterbi>
```

```
<sample realtype = "bfloat" name = "Sample" id = "sam">
  <hmm idref = "xxx"/>
</sample>
```

4.5.2.2 Code Generation

The *codeGeneration* element simply references all the algorithms defined in the previous section. The C++ source code and header files to be produced are assigned the same name as the HMM.

```
<codeGeneration file = "xxx.cc" header = "xxx.h" language = "C++">
  <forward idref = "fw"/>
  <backward idref = "bw"/>
  <viterbi idref = "vit"/>
  <sample idref = "sam"/>
</codeGeneration>
```

4.6 Model-View-Controller Architecture

The Model-View-Controller design pattern separates the code associated with the model of the problem-domain embodied in the *Model* object from the presentation code contained in the *View* object and the actions associated with the user-input in a separate *Controller* object.

The pattern of communication between the three objects is illustrated in Figure 4.3. ^[8]

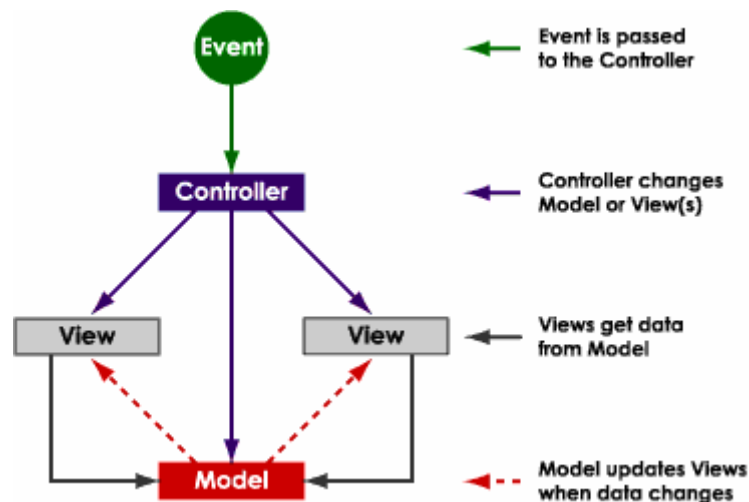


Figure 4.3 - Model-View-Controller Interaction

The Model notifies the View object of any changes in state. The View object requests updated data from the model and displays accordingly. The Controller object is used to update the Model and possibly the View object along with it as a direct result of an event.

An important thing to note is that while the View and Controller objects depend on the model, the vice versa is not the case. This separation is what makes the Model-View-Controller design pattern a popular choice and allows the Model to be designed, implemented and tested without any concern of user-interaction and presentation. Hence, it multiple views may be developed and multiple tools may be employed to provide user-input (keyboard, mouse, touch-screen etc.) for the same model without “fusing” their functionality with the underlying Model.

The above approach has been adopted in this case and a Model developed to represent a HMM. The View in this case are the Chart and the Properties and Settings of the HMM which must both be updated in the case of a change to the Model. The Controller deals with user-input in the form of direct interaction with objects on the Chart or via the widgets contained in the Properties and Settings section. This setup made it extremely simple to manage the interactions between the various components and to keep them updated with the current state of the Model. A detailed account of the implementations details are given in Section 5.5

4.7 Objected Oriented Analysis and Design

Object-Oriented Design and Analysis is a popular choice for software development as they are simple and easy to maintain. Changing the implementation of an object or adding additional functionality or components does not affect other objects as all objects are stand-alone entities. This coupled with the requirement of implementation to be carried out in Java, an object-oriented programming language led to the problem domain to be modelled as class objects.

The foremost entity that may be defined in the context of this problem is the HMM. The *HMMModel* class lies at the centre of the design of the system and encapsulates all the objects and their functions associated with them. An interface *HMMObject* has been formulated to model an entity inside the *HMMModel* class. Thus any object that wishes to be part of the HMM must implement the *HMMObject* interface.

Considering the entities that a HMM consists of leads us to a State and Transition. Multiple States linked with Transitions with suitable parameters and attributes make up a complete HMM. Hence, the State and Transitions have been abstracted as separate entities each with their unique set of attributes.

However, since both the States and Transitions may consist of one of several shapes another level of abstraction is provided in the form of *State* and *Transition* interfaces which extend the *HMMObject* interface. These are then implemented by the various State and Transition types.

An overview of the system design is presented in Appendix B, by means of a UML Class diagram, which show the main classes developed for the system and the relationship between them.

An obvious question that may arise at this stage is why implement the State and Transition objects as interfaces as opposed to abstract classes? Abstract classes allow common behaviour to be defined, while forcing their sub-classes to provide the rest. This functionality is certainly relevant in this case, as the States and Transitions while offering different shapes still have quite a few properties and operations that are similar.

The reason is simply the lack of support of the Java programming language for *Multiple Inheritance* as opposed to other object-oriented languages such as C++ that support it. Since, each implementation of the State or Transition extends a different Shape class of the Java 2D API, they cannot be made to extend another class owing to Java's limitation that a sub-class may only extend a single super-class. However, there is no limitation on the number of interfaces a class may implement and hence this approach was adopted.

Java's lack of support for multiple inheritance is believed to be as a result of the developer's goal to develop a simple programming language that uses standard concepts that most people may grasp without extensive training. In the designer's opinion, multiple inheritance causes more problems and confusion than it solves.^[4]

Chapter Five

Implementation

This chapter presents a detailed account of how the system was implemented to meet the design specification outlined in the previous chapter. A description of each component contained in G-HMMoC is presented explaining its use. Emphasis has been placed on the graphical specification and outlook features.

5.1 Java 2D

The Java 2D Application Programming Interface (API) is a set of classes that can be used to create high quality graphics. It includes features like geometric transformation, anti-aliasing, alpha compositing, image processing and bidirectional text layout, just to name a few.

The Java 2D API has been used extensively in the implementation of the drawing and editing components in this project. These include the visual representation of the State and Transition objects which are implemented as extensions of the factory classes within this category.

5.2 Interface

Section 4.2 presented a detailed design of the intended interface layout G-HMMoC which is shown in Figure 5.1 in its implemented form.

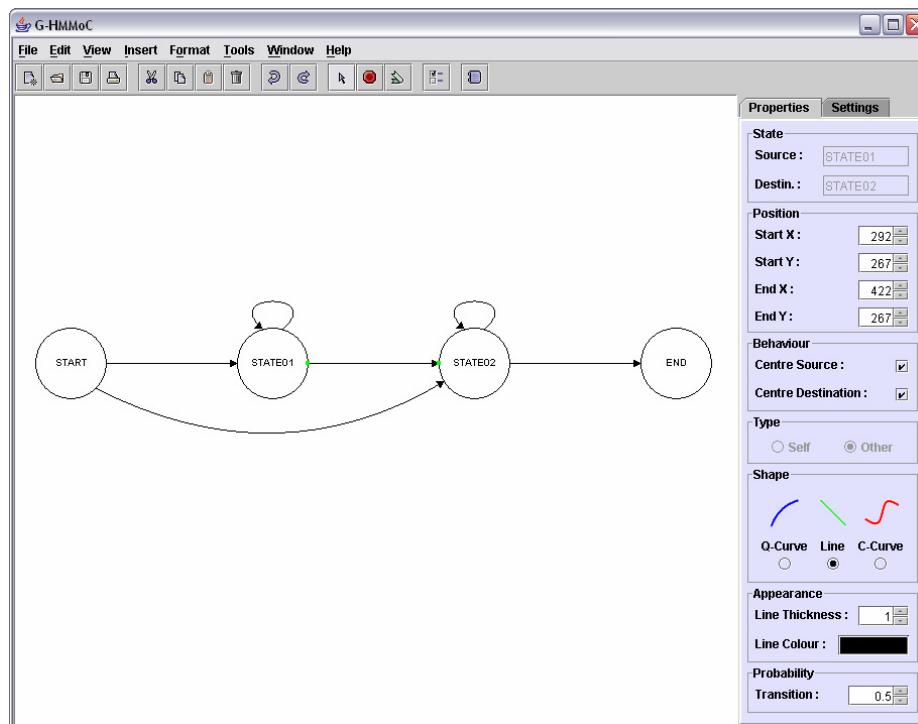


Figure 5.1 – G-HMMoC Interface Implementation

The Menus group all the functions available to the user in an appropriate category.

The Toolbar is an implementation of the *JToolbar* class of the Java Swing package and contains an array of buttons for easy access to the most frequently used functions. The use for each button is displayed as a tool-tip to the user by moving by positioning the mouse pointer over it.

The Chart is an extension of the Java AWT *JPanel* class and functions as a drawing editor. It is the main component of the application and supports insertion, deletion, selection and relocation of HMM objects.

The Properties and Settings area has been split into two distinct sections via the Java Swing *JTabbedPane* component. The *Properties* tab features the attributes of the selected object, while the *Settings* tab contains attributes applicable to the entire HMM.

A detailed description of the panels contained within each tab is given in subsequent sections.

5.3 HMM Settings

5.3.1 *Alphabet*

The *Alphabet* settings panel shown in Figure 5.2 allows the user to define an alphabet for the HMM. There is no limit to the number of characters that may be input. The text area displays up to four rows, to which a vertical scrollbar is added automatically by the system if exceeded.

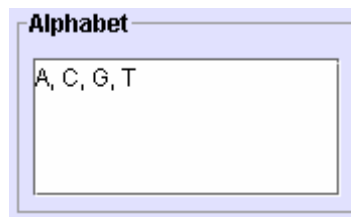


Figure 5.2 - Alphabet Settings Panel

5.3.2 *Sequences*

The *Sequences* settings panel shown in Figure 5.3 contains a spinner that allows the user to define the number of sequences emitted by the HMM. The upper and lower limits are set at 3 and 1 respectively. The spinner is enabled for use only once an alphabet has been defined.



Figure 5.3 - Sequences Settings Panel

5.3.3 Algorithms

The *Algorithms* settings panel shown in Figure 5.4 contains checkboxes to allow the user to select the applicable algorithms for a particular HMM.



Figure 5.4 - Algorithms Settings Panel

Each algorithm may be enabled or disabled with the Forward and Backward algorithms having the additional properties of generating an output table and enabling or disabling the BaumWelch, respectively. The checkboxes may be identified by moving the mouse pointer over them which displays a tool-tip to explain its use. The BaumWelch may be applied to either the Forward or Backward algorithms but not both at the same time. Thus selecting it for one algorithm automatically unselects it for the other. The options for the Output Tables are BaumWelch are only enabled if the algorithm has been selected by the user.

5.3.4 Emission Tables

Developing a practical interface for creation, editing and deletion of the Emission Probability Distribution tables proved quite difficult. A number of panels were created which together achieve the desired functionality.

Before a probability distribution may be defined, the alphabet must be set. This automatically assigns a row to each alphabet to which a value may be assigned by the user. Figure 5.5 shows a table for four alphabets using a single output tape sequence.

Symbol	Value
A	
C	
G	
T	

Figure 5.5 - Single Sequence Emission Probability Distribution Table

The system automatically adds additional rows and columns to the table if the value of the sequence parameter is incremented. Figure 5.6 shows the table for the same alphabet with pair output.

Symbol	Symbol	Value
A	A	
A	C	
A	G	
A	T	
C	A	
C	C	
C	G	
C	T	
G	A	
G	C	
G	G	
G	T	
T	A	
T	C	
T	G	
T	T	

Figure 5.6 - Double Pair Emission Probability Distribution Table

The user is only allowed to assign a probability by entering a number in the *Value* field. The value may also be defined as a fraction. The *Symbol* fields are not editable and are updated automatically, if the alphabet is changed.

Once a distribution has been defined it may be saved by assigning it a name and adding it to the list. This is carried out via the *Operation* panel as shown in Figure 5.7.

Operation

Name :

Figure 5.7 - Operation Panel for Adding/Removing Tables

The *Add* button is enabled only if a unique name has been typed which is different from the ones assigned to any distributions defined previously. The alphabet and sequences panels are disabled at this stage and may only be enabled if all distributions are removed. The *Remove* button is used to delete the selected Distribution from the list.

The *Selection* panel allows probability distributions previously defined to be selected for viewing or editing by means of a combo box as shown in Figure 5.8.

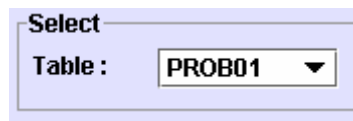


Figure 5.8 - Selection Panel for Probability Distributions

This updates the Distribution and Operation panel to contain the details of the selected distribution. Any changes made to its values are thus saved automatically.

5.4 HMM Properties

5.4.1 State

A state is created by pressing the State button on the toolbar, and clicking with the left-mouse button anywhere inside the Chart area. As long as the State button is highlighted any number of states may be inserted by a simple left-mouse click.

The state may be moved to any position within the Chart after insertion. This is achieved by selecting a single or multiple states and dragging the pointer to the desired position.

Selection of a state is done by first pressing the Select button on the toolbar and then simply pressing the left-mouse button anywhere within the state boundary. The system highlights the state as shown in Figure 5.9.



Figure 5.9 - A Selected State

Multiple selections may be made via the same manner by holding either *Shift* or *Ctrl* on the keyboard. In addition, the user may click with the left-mouse anywhere on the Chart and drag the mouse pointer to select multiple states that lie entirely within the boundary of the rectangle drawn by the system bounded between the originally clicked point and the current pointer position as shown in Figure 5.10.

The rectangle disappears as soon as the mouse is released and the bounded states are highlighted. In this case, only *State01* and *State02* will be selected and not *State03*.

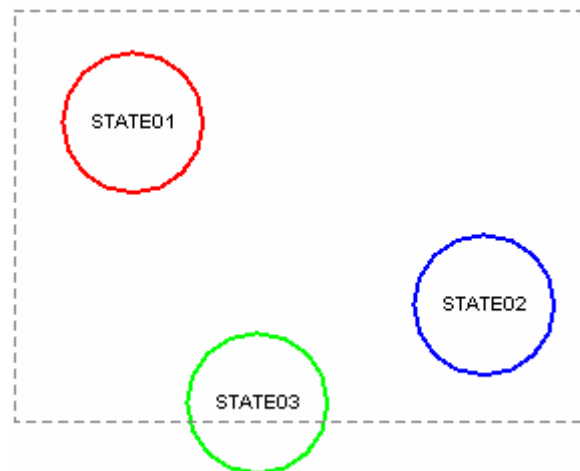


Figure 5.10 - Multiple Selections with the Mouse

The Properties of a State may be viewed and modified through the various display and control panels available in the Properties and Settings area of the interface.

5.4.1.1 General

The *General* properties panel shown in Figure 5.11 simply contains a text field to allow the user to view and modify the label of the selected state. The new label assigned after the Enter key is pressed. A maximum limit of 10 characters is enforced.

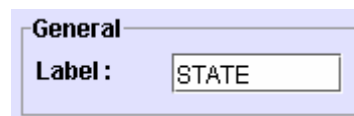
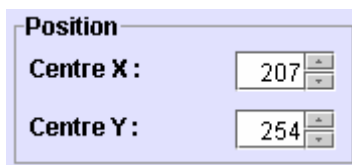


Figure 5.11 - State General Properties Panel

5.4.1.2 Position

The *Position* properties panel shown in Figure 5.12 contains spinners to allow the user to view and modify the X and Y coordinates of the centre of the State. The upper and lower input limits is constrained by the dimensions of the Chart and the size of the State.



The image shows a panel titled "Position" with two input fields. The first field is labeled "Centre X:" and contains the value "207". The second field is labeled "Centre Y:" and contains the value "254". Both fields have small up and down arrow buttons next to them, indicating they are spinners.

Figure 5.12 - State Position Properties Panel

5.4.1.3 Type

The *Type* properties panel shown in Figure 5.13 contains a set of radio buttons to allow the user to set the duty of the state in the HMM. If *Start* or *End* states are selected, the State label is automatically changed to the respective text and the State is marked internally to represent a *silent* state. Only a single Start and End state may be defined in a HMM.

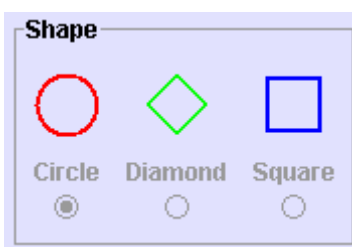


The image shows a panel titled "Type" with three radio buttons. The first is labeled "Start", the second "Other", and the third "End". The "Other" radio button is selected, indicated by a solid black dot in the center of the circle.

Figure 5.13 - State Type Properties Panel

5.4.1.4 Shape

The *Shape* properties panel shown in Figure 5.14 contains a set of radio buttons to allow the user to set the shape of the State. This feature is, at present, for display purposes only and is intended to be incorporated in the future. (See Section 8.5)



The image shows a panel titled "Shape" with three radio buttons. Each button is accompanied by a colored shape: a red circle, a green diamond, and a blue square. The labels "Circle", "Diamond", and "Square" are positioned below their respective shapes. The "Circle" radio button is selected, indicated by a solid black dot in the center of the circle.

Figure 5.14 - State Shape Properties Panel

The default shape of the state is set to Circle and is implemented as an extension of the `Ellipse2D` class of the Java 2D library.

5.4.1.5 Appearance

The *Appearance* properties panel, shown in Figure 5.15 groups the attributes relating to the visual appearance of the State.

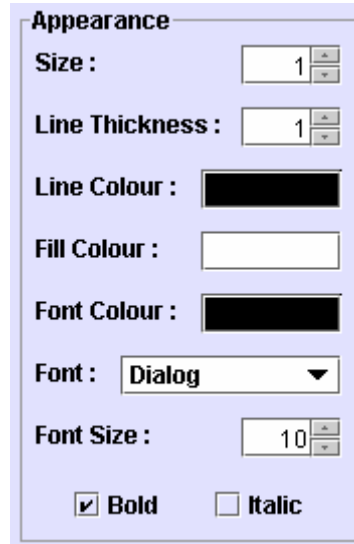


Figure 5.15 - State Appearance Properties Panel

The size and line thickness spinners are constrained with an upper limit of 30 and 10 and a lower limit of 1 and 0, respectively. The font combo box contains a list of all the fonts available locally on the system and allows the user to choose between them. The line, fill and font colours display the current settings respectively and can be modified as desired by pressing on the area with the left-mouse button. This presents the user with the dialog shown in Figure 5.16.

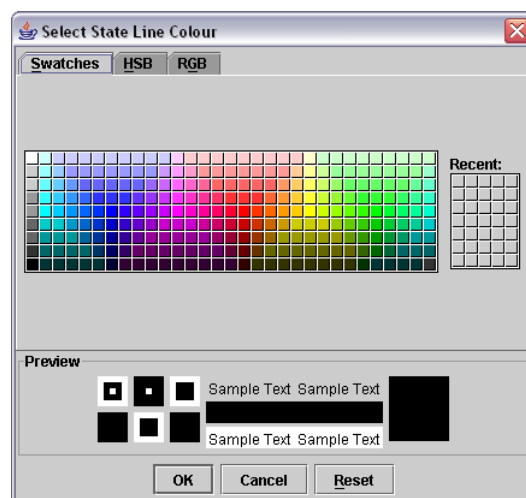


Figure 5.16 - State Colour Chooser Dialog

The appearance settings offer a means to highlight a particular State or a group a set of similar States under the same visual representation. Figure 5.17 shows a State whose appearance settings have been modified as displayed by the panel shown on the right.

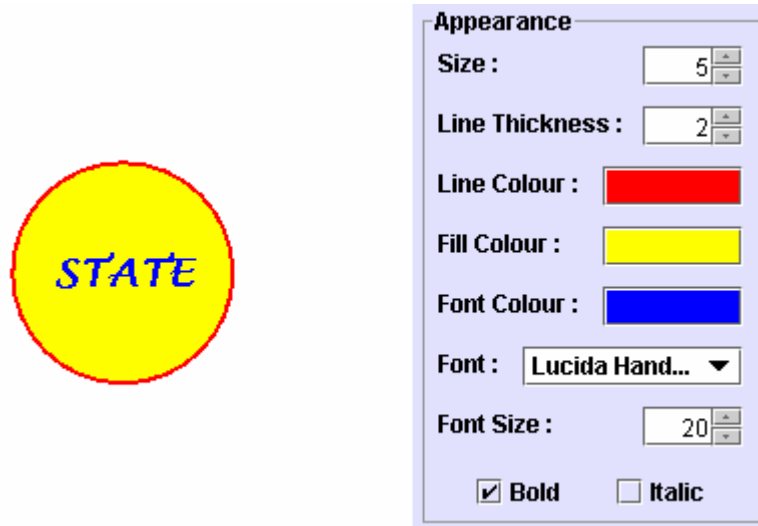


Figure 5.17 - State with modified Appearance Properties

5.4.1.6 Emission Probability

The *Emission Probability* properties panel shown in Figure 5.18 contains a combo box listing all the emission probability distribution tables that have been defined. The list is automatically updated if a new table is added or an existing one removed.

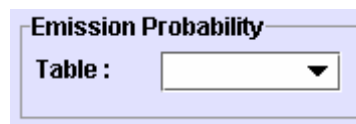


Figure 5.18 - State Emission Probability Properties Panel

5.4.2 Transition

A transition is created by pressing the Transition button on the toolbar, and clicking with the left-mouse button on any two states. As long as the Transition button is highlighted any number of transitions may be inserted in the same manner.

The transition is drawn with its end points placed on the boundary of the circle as shown in Figure 5.19.



Figure 5.19 - A Transition between two States

The transition may be moved to any position owing to its behaviour along the boundary of its source and destination states. This is achieved by selecting the relevant end point of the transition and dragging the pointer to the desired position.

The selection of a transition is done in the same manner as a State, as explained in Section 5.4.1. The system highlights its end and control points (if applicable) and draws tangents as dotted lines from the control points to the end points. This is illustrated in Figure 5.20 for a Quadratic and Cubic Curve Transition.

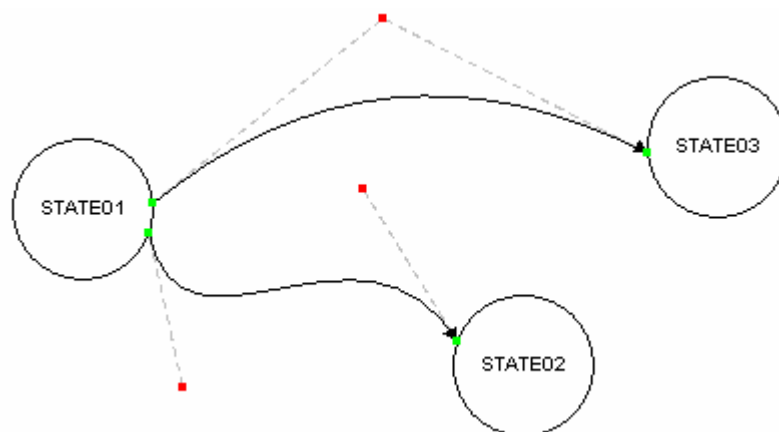


Figure 5.20 - Selected Quadratic and Cubic Curve Transitions

The direction of the Transition is marked by an arrow drawn at the Destination State end pointing towards it. The arrow is redrawn if the State or Transition is moved, to point in the appropriate direction. This is simply the direction of the straight line in case of a Line Transition and a tangent from the control to the end point for Curves.

The Properties of a Transition may be viewed and modified through the various display and control panels available in the Properties and Settings area of the interface.

5.4.2.1 State

The *State* properties panel shown in Figure 5.21 simply contains a text field to allow the user to only view the labels of the selected transition's source and destination states.

Figure 5.21 - Transition State Properties Panel

5.4.2.2 Position

The *Position* properties panel contains spinners to allow the user to view and modify the X and Y coordinates of the end and control points of a Transition. The upper and lower input limits of the end points are constrained with respect to the boundary of the transition's source and destination states respectively. The control points may be set to any value within the dimensions of the Chart.

The control points are only applicable to Curve Transitions and hence only relevant information is displayed. Figure 5.22 shows the position properties panels for Line, Quadratic Curve and Cubic Curve Transitions respectively.

Figure 5.22 - Transition Position Properties Panels

5.4.2.3 Behaviour

The *Behaviour* properties panel shown in Figure 5.23 contains check boxes to allow the user to lock the transition end points to the centre of its source and destination states. Once unlocked, the points may be moved or positioned anywhere along the state boundary. This helps in achieving a neat and desirable layout of the HMM. This option is disabled for a self transition.

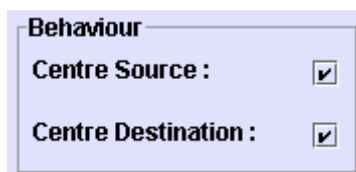


Figure 5.23 - Transition Behaviour Properties Panel

5.4.2.4 Type

The *Type* properties panel shown in Figure 5.24 contains a set of radio buttons (intended for viewing purposes only) to display the type of the selected transition.



Figure 5.24 - Transition Type Properties Panel

A self transition is classified as having the same source and destination states. It is drawn automatically by the system by clicking with the left-mouse button twice on the same state, provided it does not already contain one. A self-transition is an instance of a *CubicCurveTransition* and its shape category cannot be changed although the end and control points may be positioned just like any other transition. An example of a state with a default self transition with control and end points highlighted due to selection is shown in Figure 5.25.

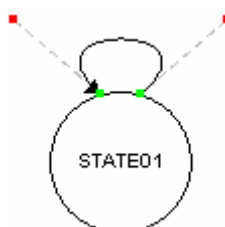


Figure 5.25 - State with Self Transition

5.4.2.5 Shape

The *Shape* properties panel shown in Figure 5.26 contains a set of radio buttons to allow the user to set the shape of the Transition. The default is set to a straight line with the choice of a Quadratic Curve with a single control point or a Cubic Curve with two control points. This offers extreme flexibility in achieving a neat and desirable layout of the HMM.

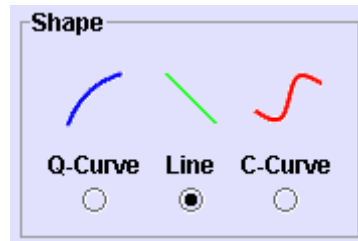


Figure 5.26 - Transition Shape Properties Panel

The Line, Quadratic Curve and Cubic Curve Transitions are simply implemented as extensions of the *Line2D*, *QuadCurve2D* and *CubicCurve2D* classes of the Java 2D API respectively. Additional functionality has been added to include the attributes required to operate as part of the overall HMM architecture. An example of a graph with states interconnected by means of a transition of each category is shown in Figure 5.27.

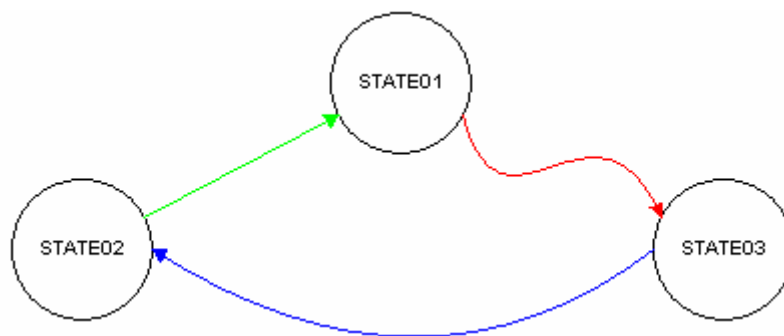


Figure 5.27 - Line, Quadratic Curve and Cubic Curve Transitions

5.4.2.6 Appearance

The *Appearance* properties panel shown in Figure 5.28 collates the attributes relating to the visual appearance of the Transition.



Figure 5.28 - Transition Appearance Properties Panel

The line thickness spinner is constrained with an upper limit of 3 and a lower limit of 0 makes draws it dotted. The line colour displays the current colour of the line that may be modified in the same manner as the state in Section 5.4.1.5.

5.4.2.7 Probability

The *Probability* properties panel shown in Figure 5.29 contains a spinner contained with an upper limit of 1 and a lower limit of 0 to allow the user to set a probability value for the selected transition.

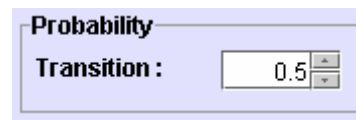


Figure 5.29 - Transition Probability Properties Panel

5.5 Observer/Observable Design Pattern

Section 4.6 discussed the use of the Model-View-Controller architecture to handle the communication between the various objects in use. Java provides a simple mechanism to achieve this functionality in the form of the Observer/Observable design pattern.

The *Observable* object is the equivalent of the Model and can have any number of Views, called *Observers*. When an Observable object changes, it notifies all Observers that have registered their interest in it to be updated. The mode of communication that has been implemented for the classes for G-HMMoC is represented in Figure 5.30.

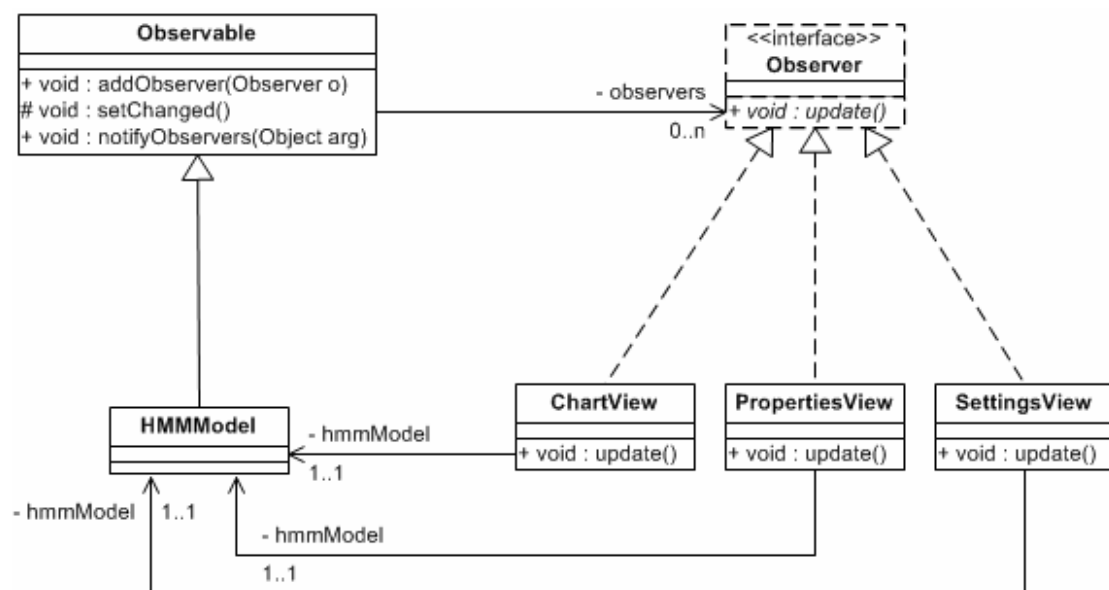


Figure 5.30 - Observer/Observable Design Pattern for G-HMMoC

Chapter Six

Evaluation

The system was evaluated periodically during the design and implementation stages and any deviations found were immediately rectified. This chapter presents an evaluative study of the operation of G-HMMoC by modelling an example of the Occasionally Dishonest Casino. The complete scenario is explained in [6] as follows:

In a Casino they use a fair die most of the time, but occasionally they switch to a loaded die. The loaded die has probability 0.5 of a six and a probability 0.1 for the numbers one to five. Assume that the casino switches from a fair to a loaded die with probability 0.05 before each roll, and that the probability of switching back is 0.1. Then the switch between dice is a Markov process. In each state of the Markov process the outcome of a roll have different probabilities, and thus the whole process is an example of a Hidden Markov Model. We can draw it like this:

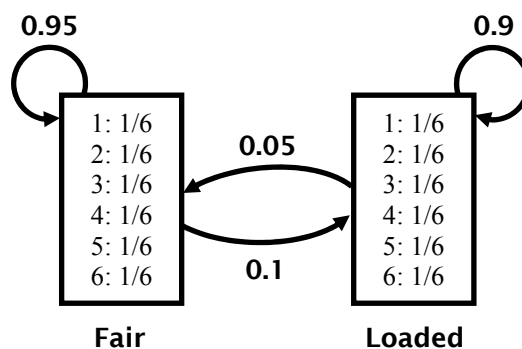


Figure 6.1 - Fair and Loaded Die

Where the emission probabilities $e()$ are shown in the state boxes.

The process of representing the above HMM in G-HMMoC is now discussed. Start and End States have been added to the above with the assumption that the process always begins with a fair die but may end with either (which is traditionally the case with Markov chains). However, the probability of the transitions from either states to the End State has been set to zero, which implies that the process never ends (an unrealistic expectation), but the emphasis is on demonstrating the capabilities of G-HMMoC for modelling the problem and producing an equivalent XML representation to serve as input for HMMoC.

6.1 HMM Settings

Before designing the HMM layout, the settings applicable to the HMM globally must be defined.

6.1.1 *Alphabet*

The alphabet of a HMM is the set of output symbols it may generate which in this case of a single die are simply the numbers 1 to 6. These are added to the alphabet panel separated by commas as shown in Figure 6.2.

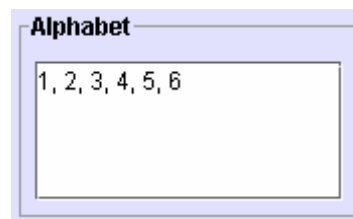


Figure 6.2 - Defining Alphabets for Casino

6.1.2 *Sequences*

The value for this is kept at the default setting of 1 and with a single die; only one set of symbol is emitted on each roll.

6.1.3 *Algorithms*

The relevant algorithms may be chosen by the user depending on the result to be computed inside HMMoC. This is outside the scope of this project and hence not explored furthered. It is sufficient to note that the interface is flexible enough to allow for any combination to be set.

6.1.4 *Emission Tables*

The emission probability values associated with each state are defined separately and assigned a unique name. The state is then set to reference the relevant table. Although not applicable in this example, this may be particularly useful where there is a set of states with the same probability distributions, thereby saving the user from having to input them individually for each state. Figure 6.3 shows the tables defined for the fair and loaded states respectively.

Select	
Table : FAIR	
Distribution	
Symbol	Value
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

Select	
Table : LOADED	
Distribution	
Symbol	Value
1	1/10
2	1/10
3	1/10
4	1/10
5	1/10
6	1/2

Figure 6.3 - Emission Probabilities for Casino

6.2 HMM Drawing

The HMM is drawn by inserting the desired number of states in the chart and connecting them by specifying the transitions that apply. The features of G-HMMoC that are relevant to this task have been explained in detail in Chapter 5. The final version, of the HMM for Casino, is shown in Figure 6.4.

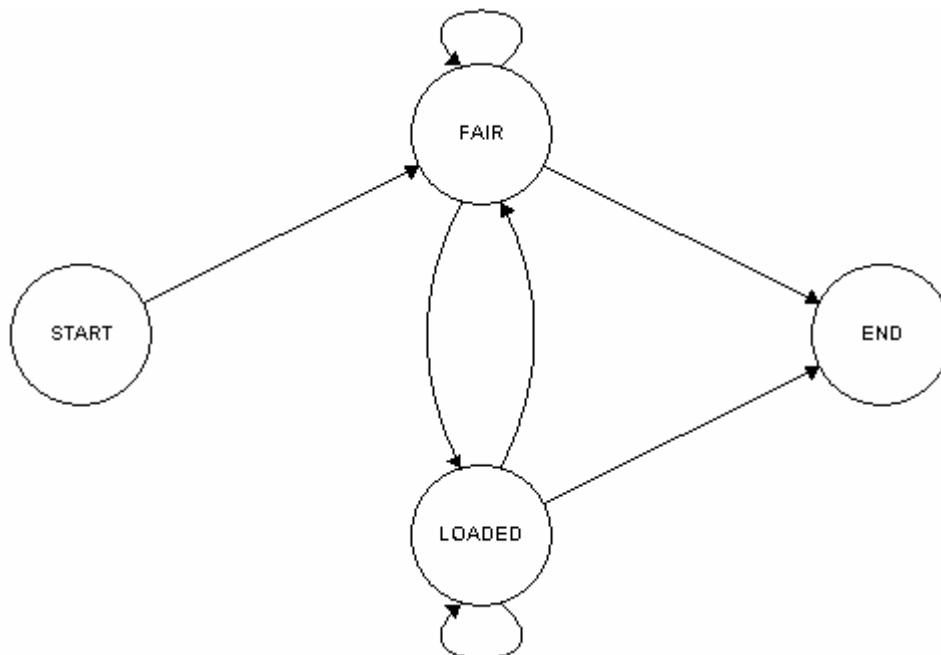


Figure 6.4 - HMM for Casino

All objects are initially inserted into the chart with their default properties. These may be changed by the user to achieve the desired representation and layout and assign the relevant properties. This process is discussed in the next section.

6.3 HMM Object Properties

While the properties of some of the objects in the HMM in Figure 6.4 have not been changed from their default values, others were modified to achieve the layout shown. The States are aligned with respect to their X and Y coordinates to give a symmetrical look. This is possible by positional the objects manually using the mouse but absolute accuracy is possible by changing the values via the Properties Panel. Figure 6.5 shows the values of the X coordinates of the fair and loaded states respectively, have been set to the same value to achieve a perfect alignment along the y-axis.



Figure 6.5 - Position Properties for Fair and Loaded States

The same technique was used for the Transitions connecting the Fair and Loaded states to appear symmetrical by setting their control points to be equally distant from their end points.

As the default shape of a transition inserted by G-HMMoC is set to *Line*, the shape property of these transitions was changed using the Shape Panel.

6.4 Save to File

The graphical representation of the HMM for Casino, may be saved to file and retrieved later. A detailed account of this process has been described previously in Section 4.4 The textual output produced to save the Casino HMM is given in Appendix C for illustration purposes to demonstrate its correctness. This process is completely transparent to the user and meant for use by the system only.

6.5 Generate XML

Once, the specification of the HMM is complete, it may be translated into equivalent XML representation recognised by HMMoC. This is output to a file with the same name assigned to the G-HMMoC, with an “.xml” extension.

The XML output produced for the HMM in Figure 6.4 is now presented.

```

<?xml version = "1.0"?>
<author> G-HMMoC </author>

<hml>
  <alphabet id = "alphabets">
    123456
  </alphabet>

  <output id = "sequenceX">
    <alphabet idref = "alphabets"/>
    <identifier type = "sequence" value = "aseqX"/>
    <identifier type = "length" value = "iLenX"/>
    <code type = "parameter" value = "char* aSeqX"/>
    <code type = "parameter" value = "int iLenX"/>
  </output>

  <hmm id="Casino">

    <outputs id = "hmmoutputs">
      <output idref = "sequenceX"/>
    </output>

    <block id = "block1">
      <state id = "START"/>
    </block>

    <block id = "block2">
      <state id = "FAIR" emission = "FAIR"/>
      <state id = "LOADED" emission = "LOADED"/>
    </block>

    <block id = "block3">
      <state id = "END"/>
    </block>

    <graph>
      <block idref = "block1"/>
      <block idref = "block2"/>
      <block idref = "block3"/>
    </graph>

    <transitions>
      <transition from = "START" to = "FAIR" probability = "t1"/>
      <transition from = "FAIR" to = "FAIR" probability = "t2"/>
      <transition from = "FAIR" to = "LOADED" probability = "t3"/>
      <transition from = "LOADED" to = "LOADED" probability = "t4"/>
      <transition from = "LOADED" to = "FAIR" probability = "t5"/>
      <transition from = "FAIR" to = "END" probability = "t6"/>
      <transition from = "LOADED" to = "END" probability = "t7"/>
    </transitions>

```

```

<probability id = "t1">
  <code type = "expression"> 1.0 </code>
</probability>
<probability id = "t2">
  <code type = "expression"> 0.95 </code>
</probability>
<probability id = "t3">
  <code type = "expression"> 0.05 </code>
</probability>
<probability id = "t4">
  <code type = "expression"> 0.9 </code>
</probability>
<probability id = "t5">
  <code type = "expression"> 0.1 </code>
</probability>
<probability id = "t6">
  <code type = "expression"> 0.0 </code>
</probability>
<probability id = "t7">
  <code type = "expression"> 0.0 </code>
</probability>

<emission id = "empty">
  <probability>
    <code type = "expression"> 1.0 </code>
  </probability>
</emission>

<emission id = "FAIR">
  <output idref="sequenceX"/>
  <probability>
    <code type="statement">
      <identifier output = "sequenceX"
        value = "iEmissionX"/>
      <identifier type = "result" value = "iProb"/>
      <![CDATA[
        switch(iEmissionX) {
          case '1': iProb = 1.0/6.0; break;
          case '2': iProb = 1.0/6.0; break;
          case '3': iProb = 1.0/6.0; break;
          case '4': iProb = 1.0/6.0; break;
          case '5': iProb = 1.0/6.0; break;
          case '6': iProb = 1.0/6.0; break;
          default: cerr << "Emission Table Error" <<
            endl;
        }
      ]]>
    </code>
  </probability>
</emission>

```

```

<emission id = "LOADED">
  <output idref="sequenceX"/>
  <probability>
    <code type="statement">
      <identifier output = "sequenceX"
        value = "iEmissionX"/>
      <identifier type = "result" value = "iProb"/>
      <![CDATA[
        switch(iEmissionX) {
          case '1': iProb = 1.0/10.0; break;
          case '2': iProb = 1.0/10.0; break;
          case '3': iProb = 1.0/10.0; break;
          case '4': iProb = 1.0/10.0; break;
          case '5': iProb = 1.0/10.0; break;
          case '6': iProb = 1.0/2.0; break;
          default: cerr << "Emission Table Error" <<
            endl;
        }
      ]]>
    </code>
  </probability>
</emission>

</hmm>

<forward realtype = "bfloat" outputTable = "yes"
  baumWelch = "no" name = "Forward" id = "fw">
  <hmm idref = "Casino"/>
</forward>

<backward realtype = "bfloat" outputTable = "yes"
  baumWelch = "no" name = "Backward" id = "bw">
  <hmm idref = "Casino"/>
</backward>

<viterbi realtype = "bfloat" name = "viterbi" id = "vit">
  <hmm idref = "Casino"/>
</viterbi>

<sample realtype = "bfloat" name = "sample" id = "sam">
  <hmm idref = "Casino"/>
</sample>

<codeGeneration file = "Casino.cc" header = "Casino.h"
  language = "C++">
  <forward idref = "fw"/>
  <backward idref = "bw"/>
  <viterbi idref = "vit"/>
  <sample idref = "sam"/>
</codeGeneration>

</html>

```

Chapter Seven

Conclusion

HMMoC translates a high-level description of an arbitrary HMM into C++ code for computation of likelihoods, posterior probabilities, path decoding and training. The HMM is specified textually using XML representation according to the syntactic and semantic rules governing the process. This approach although extremely successful had a number of drawbacks which limited its popularity, the prime one being that it required the user to learn a sizable set of rules of specification for HMMoC in XML format. Further, this is far from the natural way of specifying HMMs which is to draw a graph representing the Markov chain and specify the associated transition and emission probabilities.

This project was hence undertaken to combat this shortcoming and provide a graphical layer between the user and the XML specification. A theoretical understanding of HMMs and their use was established, which offered a useful insight of the features that are essential for an accurate modelling of HMMs. This was coupled with extensive study of input interface of HMM and probing of its developers to arrive at a conclusive set of requirements for the project.

Since, the aim of the project was to deliver a complete working program and not an experimental one, proven software development methodologies, learnt in various courses during the MSc, were employed. This entailed performing a detailed Use Case Analysis to gather requirements and using the Prototyping Model to refine and perfect them. Once finalised, a formal system design was formulated and presented using a UML Class Diagram to show the static structure of the classes and their dependencies. In addition, a detailed analysis of the syntactic and semantic rules of specification governing the input to HMMoC was carried out to yield a small subset of XML elements that the graph representation may easily be translated into.

The implementation of G-HMMoC was simplified with the use of the Java 2D API, which contains advanced visual outlook and drawing features. Prior to the development of Java 2D API, Java had very limited support for the advanced operations that were needed to achieve a graphical representation of a HMM.

A particular mention in this context must also be made about the Model View Controller design pattern that was implemented using the Java factory offered *Observable* class and *Observer* interface. This approach separated the details surrounding the modelling of the HMM from the presentation making the addition of multiple views such as the *Chart*, *Properties* and *Settings* straight forward. The Observable/Observer design pattern provided a transparent communication between the various components, once registered to receive the relevant notifications. This further simplified the overall design and coding of the system.

Finally, an evaluative study on the operation of G-HMMoC was presented to model the behaviour of the *Occasionally Dishonest Casino*. The application proved successful in modelling all aspects of the HMM and achieving an aesthetically pleasing graphical layout. The XML output produced was analysed against the set of rules that are parsed successfully by HMMoC and found to be accurate.

Hence, it can be concluded, that G-HMMoC has successfully managed to achieve all the objectives it was proposed to meet. Flexibility was achieved by offering the user to specify the number sequences output by the HMM, which makes the application capable of modelling pair HMMs and triple HMMs.

In addition, although G-HMMoC was meant to cater for the desired task it was designed to be capable of modelling HMMs across various disciplines and also prove as a valuable tool for understanding and the teaching of the concepts surrounding HMMs. Finally, it may also be utilised as a “fancy” editor to draw complex graphs or simply modelling data flow diagrams.

Chapter Eight

Future Development

Although the project has been successful in achieving its primary objectives there are many avenues for further work, ranging from the improvement of existing functionality to the incorporation of new features. This chapter proposes the possibility of addition of the most desirable amongst these.

8.1 Chart Settings

The Chart area of G-HMMoC is fairly limited and only provides support for basic editing. In addition, the drawing is limited with respect to the dimensions of the application. A richer set of features should be offered such as the ability to the set the dimensions of the Chart and using horizontal and/or vertical scrollbars to navigate to the desired display.

Further, the functionality to display a grid in the background and an option to make the objects snap to the grid settings is a standard setting across the majority of drawing tools. It is believed that these features will make the alignment of the objects simpler.

8.2 Multiple Selected Objects Properties

The Properties panel currently displays the attributes when a single State or Transition is selected. In the case, that the user intends to set an attribute for a group of objects, one will have to do so individually which may prove to be quite tedious. It is therefore proposed that the common settings of multiple selected objects be displayed if selected and any property change applied to the entire selection.

8.3 Clipboard Support

The system currently provides no support for cut/copy/paste operations. This functionality is extremely desirable in all graphical editing environments and may be included in G-HMMoC to enhance the editing capabilities further.

8.4 Consistency Check

A Consistency Check in this context simply implies checking the semantics of the HMM specification. These may include constraints such as the sum of all transitions emerging from a State must sum to one. This will prevent errors on the part of the user during specification to be caught early and from being propagated to HMMoC.

8.5 State Shapes

This feature has been discussed previously in Section 5.4.1.4. Different States are used particularly in profile HMMs to represent States intended for various purposes (insertion/deletion etc.). This feature should be incorporated to model these effectively.

8.6 Repetitive Structures

Repetitive structures within the graph specifying the HMM are fairly common amongst profile HMMs. Currently there is no support to model these effectively. The addition of the clipboard will lend some help towards this but the ideal way to achieve this may be by drawing a graph of states and setting a parameter to indicate the number of repetitions that apply to it.

8.7 One-Touch Compilation

The current overall process of specifying the HMM in G-HMMoC, generating XML input for HMMoC and then executing the program is quite cumbersome and spread across multiple applications. It is highly desirable to merge this process and offer a *one-touch* compilation directly to C++ code using HMMoC directly from G-HMMoC.

8.8 Results

This is an extension to the previous proposal. The idea is to represent the output from the execution of the generated program to be brought back into G-HMMoC and displayed in the most meaningful manner to the user.

Appendix A

Use Case Analysis

A.1 Define Alphabet

Summary	A set of alphabet must be defined for the HMM prior to specification.
Actors	System User.
Priority	Must have. The ability to allow users to define an algorithm essential to construct a HMM and specify emission probability distributions for States.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User enters the Alphabet as a string of characters separated by commas, space or both.2. System parses the input and updates the Probability Distribution Table to contain a row per Alphabet.
Sub Cases	None.
Extensions	None.

A.2 Specify Sequences

Summary	The number of output tapes emitted by the HMM may be defined as a variable parameter to increase the functionality and usefulness of the application.
Actors	System User.
Priority	Should have. The addition of this functionality will allow Pair-HMMs to be specified.
Preconditions	None.
Flow of Events	1. User selects the number of sequences to be output by the HMM between 1 and 3.
Sub Cases	None.
Extensions	<i>Select Multiple States</i>

A.3 Specify Algorithms

Summary	The XML specification includes the algorithms that are applied during the execution of the generated program.
Actors	System User.
Priority	Must have. The selection of applicable algorithms is essential to achieve the desired XML output.
Preconditions	None.
Flow of Events	1. User chooses the algorithm to be used from the following choices: <ul style="list-style-type: none">i. Forwardii. Backwardiii. Viterbiiv. Sample 2. User chooses whether an output table and baumWelch apply to the Forward or Backward Algorithms.
Sub Cases	None.
Extensions	

A.4 Define Probability Distribution

Summary	The user must be able to define a Probability Distribution for assignment to States.
Actors	System User.
Priority	Must have. The ability to allow users to define an Emission Probability Distribution is essential to construct a HMM.
Preconditions	<ol style="list-style-type: none"> 1. Alphabet must be defined. (<i>See Define Alphabet</i>) 2. Number of Sequences must be specified. (<i>See Specify Sequences</i>)
Flow of Events	<ol style="list-style-type: none"> 1. User assigns a name to the Probability Distribution. 2. User adds the Distribution to the List of Distributions by pressing the “Add” button. 3. User enters the respective probabilities associated with each Emission Alphabet and presses the <i>Enter Key</i>.
Sub Cases	None.
Extensions	None.

A.5 Select Probability Distribution

Summary	The user must be able to select a Probability Distribution for viewing, editing or deletion.
Actors	System User.
Priority	Must have. The ability to allow users to select an Emission Probability Distribution is essential to view or edit them.
Preconditions	<ol style="list-style-type: none"> 1. Probability Distribution must be defined. (<i>See Define Probability Distribution</i>)
Flow of Events	<ol style="list-style-type: none"> 1. User selects the desired Probability Distribution from the list of already defined Distributions. 2. System loads the contents of the Distribution in the Probability Table for display.
Sub Cases	None.
Extensions	None.

A.6 Delete Probability Distribution

Summary	The user must be able to delete and already defined Probability Distribution.
Actors	System User.
Priority	Must have. The ability to allow users to delete an Emission Probability Distribution is essential to maintain the desired set of Distribution for the HMM.
Preconditions	1. Probability Distribution must be selected. (See <i>Select Probability Distribution</i>)
Flow of Events	1. User removes the Distribution by pressing the “Remove” button. 2. System deletes the selected Distribution from the List of Distributions.
Sub Cases	None.
Extensions	None.

A.7 Insert State

Summary	A HMM is composed of several states linked together. The user must be able to insert a State at any position on the chart.
Actors	System User.
Priority	Must have. The ability to allow users to insert states is essential to construct a HMM.
Preconditions	None.
Flow of Events	1. User selects the <i>Insert State</i> Option from the Toolbar. 2. User clicks with the left-mouse button anywhere on the Chart. 3. System inserts a State with default properties with the clicked point as centre.
Sub Cases	None.
Extensions	None.

A.8 Select Single State

Summary	The user must be able to select any State of the HMM on the chart.
Actors	System User.
Priority	Must have. The ability to allow users to select States is essential to be able to manipulate them.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Select</i> Option from the Toolbar.2. User clicks with the left-mouse button anywhere inside the State boundary.3. System clears any existing selections.4. System selects the State and indicates so by highlighting it on the Chart.5. System displays the Properties for the State.
Sub Cases	None.
Extensions	<i>Select Multiple States</i>

A.9 Select Multiple States

Summary	The user should be able to several States of the HMM on the chart for collective positioning or otherwise.
Actors	System User.
Priority	Should have. The ability to allow users to select multiple States is not essential but will make it easier to carry out certain tasks.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Select</i> Option from the Toolbar.2. User holds down the <i>Shift</i> or <i>Ctrl</i> Key on the Keyboard.3. User clicks with the left-mouse button anywhere inside the State boundary.4. System selects the State and indicates so by highlighting it on the Chart.5. System clears the Properties display.
Sub Cases	None.
Extensions	<ol style="list-style-type: none">1. The User may also choose the <i>Select</i> Option also click and drag using left-mouse button anywhere on the Chart.2. The System draws a dotted rectangle with respect to the clicked position and the current position.3. The System selects all States that lie within the bounds of the <i>Selection Rectangle</i>.

A.10 Change State Label

Summary	The user must be able to change State Label to make it unique for identification.
Actors	System User.
Priority	Must have. The ability to allow users to change State Label is essential to make each State in the HMM unique.
Preconditions	1. State must be Selected. (<i>See Select Single State</i>)
Flow of Events	1. User types the new State Label in the Properties Panel and presses Enter. 2. System display the new Label on the State.
Sub Cases	None.
Extensions	None.

A.11 Change State Type

Summary	The user should be able to change the State Type to identify its use to the System.
Actors	System User.
Priority	Should have. The ability to allow users to change State Type indicates its use and constraints to the System.
Preconditions	1. State must be Selected. (<i>See Select Single State</i>)
Flow of Events	1. User chooses the desired Type from the following three options. <ol style="list-style-type: none"> i. Start (<i>One per HMM</i>) ii. Other (<i>All other States</i>) iii. End (<i>One per HMM</i>)
Sub Cases	None.
Extensions	None.

A.12 Change State Appearance

Summary	The user may be able to change the appearance of a State in order to highlight attention.
Actors	System User.
Priority	Nice to have. This functionality is not essential to the operation of the HMM and may be included if time permits.
Preconditions	1. State must be Selected. <i>(See Select Single State)</i>
Flow of Events	1. User chooses the desired Appearance Setting from amongst the following options. <ol style="list-style-type: none"> i. Size <i>(Sets the display size of the State)</i> ii. Line Thickness <i>(Sets the thickness of the Outline)</i> iii. Line Colour <i>(Sets the colour of the Outline)</i> iv. Fill Colour <i>(Sets the colour of the Background)</i> v. Font Colour <i>(Sets the colour of the State Label)</i> vi. Font <i>(Sets the Font of the State Label)</i> vii. Font Size <i>(Sets the Font size of the State Label)</i> viii. Bold/Italic <i>(Sets the State Label to Bold/Italic)</i>
Sub Cases	None.
Extensions	None.

A.13 Assign Emission Probability Distribution

Summary	The user must be able to assign a Probability Distribution to a State to construct a HMM.
Actors	System User.
Priority	Must have. The ability to allow users to assign an Emission Probability Distribution is essential to construct a HMM.
Preconditions	1. State must be Selected. <i>(See Select Single State)</i> 2. Emission Probability Distribution must be defined. <i>(See Define Probability Distribution)</i> .
Flow of Events	1. User chooses the desired Distribution from amongst the ones defined.
Sub Cases	None.
Extensions	None.

A.14 Change State Shape

Summary	The user may be able to change State Shape to construct HMMs where different groups of States may be identified.
Actors	System User.
Priority	Nice to have. This functionality is not essential to the operation of the HMM and may be included if time permits.
Preconditions	1. State must be Selected. (<i>See Select Single State</i>)
Flow of Events	1. User chooses the desired Shape from the following three options. <ol style="list-style-type: none">i. Circle (<i>Default</i>)ii. Diamondiii. Square
Sub Cases	None.
Extensions	None.

A.15 Insert Transition

Summary	A Transition may be inserted between any two States in order to link them to construct a HMM.
Actors	System User.
Priority	Must have. The ability to allow users to specify Transitions between States is essential to construct a HMM.
Preconditions	None.
Flow of Events	1. User selects the “Insert Transition” Option from the Toolbar. 2. User clicks with the left-mouse button on any State. 3. System selects the State as Source. 4. User clicks with the left-mouse button on another State. 5. System selects the State as Destination 6. System inserts a Transition with default properties between the two States.
Sub Cases	None.
Extensions	1. If the Source and Destination States are the same then a Self-Transition must be inserted.

A.16 Move State

Summary	In order to achieve the best layout for the HMM, the user must be able to move a State to any position on the chart.
Actors	System User.
Priority	Must have. The ability to allow users to position States is essential to achieve a desirable layout of the HMM.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none"> 1. User clicks on State and drags the pointer using the left-mouse button. 2. System selects State and places it at the desired position.
Sub Cases	None.
Extensions	<ol style="list-style-type: none"> 1. If Multiple States are selected then all Selected States are moved relative to their original position. 2. If Transitions exists for the State then the System positions all Transitions relative to the moved State.

A.17 Delete State

Summary	The user may wish to delete a State from the HMM.
Actors	System User.
Priority	Must have. The ability to allow users to delete States is essential to achieve a desirable layout of the HMM.
Preconditions	<ol style="list-style-type: none"> 1. State must be Selected. (See <i>Select Single State / Select Multiple States</i>)
Flow of Events	<ol style="list-style-type: none"> 1. User pressed <i>Del</i> Key on the Keyboard. 2. System presents a dialog to confirm to proceed or cancel. 3. If User chooses to proceed, then System deletes Selected State.
Sub Cases	None.
Extensions	<ol style="list-style-type: none"> 1. If Multiple States are selected then all Selected States are deleted. 2. If Transitions exists for the State then the System deletes all Transitions linked to the State. (See <i>Delete Transition</i>)

A.18 Select Single Transition

Summary	The user must be able to select any Transition of the HMM on the chart.
Actors	System User.
Priority	Must have. The ability to allow users to select Transitions is essential to be able to manipulate them.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Select</i> Option from the Toolbar.2. User clicks with the left-mouse button anywhere on the Transition.3. System clears any existing selections.4. System selects the Transition and indicates so by highlighting it on the Chart.5. System displays the Properties for the Transition.
Sub Cases	None.
Extensions	<i>Select Multiple Transitions</i>

A.19 Select Multiple Transitions

Summary	The user should be able to several Transitions of the HMM on the chart for collective manipulation.
Actors	System User.
Priority	Should have. The ability to allow users to select multiple Transitions is not essential but will make it easier to carry out certain tasks.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Select</i> Option from the Toolbar.2. User holds down the <i>Shift</i> or <i>Ctrl</i> Key on the Keyboard.3. User clicks with the left-mouse button anywhere on the Transition.4. System selects the Transition and indicates so by highlighting it on the Chart.5. System clears the Properties display.
Sub Cases	None.
Extensions	<ol style="list-style-type: none">1. The User may also choose the <i>Select</i> Option also click and drag using left-mouse button anywhere on the Chart.2. The System draws a dotted rectangle with respect to the clicked position and the current position.3. The System selects all Transitions that lie within the bounds of the <i>Selection Rectangle</i>.

A.20 Delete Transition

Summary	The user may wish to delete a Transition from the HMM.
Actors	System User.
Priority	Must have. The ability to allow users to delete Transitions is essential to achieve a desirable layout of the HMM.
Preconditions	1. Transition must be Selected. <i>(See Select Single Transition / Select Multiple Transitions)</i>
Flow of Events	1. User pressed <i>Del</i> Key on the Keyboard. 2. System presents a dialog to confirm to proceed or cancel. 3. If User chooses to proceed, then System deletes Selected Transition.
Sub Cases	None.
Extensions	1. If Multiple Transitions are selected then all Selected Transitions are deleted.

A.21 Set Transition Probability

Summary	The user must be able to set the Probability of a Transition.
Actors	System User.
Priority	Must have. The ability to allow users to set a Probability for each Transition is essential to construct a complete HMM.
Preconditions	1. Transition must be Selected. <i>(See Select Single Transition)</i>
Flow of Events	1. User assigns a number between 0 and 1 to the Transition.
Sub Cases	None.
Extensions	None.

A.22 Move Transition

Summary	In order to achieve the best layout for the HMM, the user must be able to move a Transition to any position on the chart with respect to the State.
Actors	System User.
Priority	Should have. The ability to allow users to position Transitions with respect to its Source and Destination States is important to achieve a desirable layout of the HMM.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User clicks on Transition end points and drags the pointer using the left-mouse button.2. System selects Transition and places it at the desired position with respect to the Source or Destination State boundary.
Sub Cases	None.
Extensions	<ol style="list-style-type: none">1. If Multiple States are selected then all Selected States are moved relative to their original position.2. If Transitions exists for the State then the System positions all Transitions relative to the moved State.

A.23 Change Transition Shape

Summary	The user may be able to change Transition Shape to construct HMMs with a neat layout.
Actors	System User.
Priority	Nice to have. This functionality is not essential to the operation of the HMM and may be included if time permits.
Preconditions	1. Transition must be Selected. (<i>See Select Single Transition</i>)
Flow of Events	1. User chooses the desired Shape from the following three options. <ol style="list-style-type: none">i. Quad-Curveii. Line (<i>Default</i>)iii. Cubic-Curve
Sub Cases	None.
Extensions	None.

A.24 Change Transition Appearance

Summary	The user may be able to change the appearance of a Transition in order to highlight attention.
Actors	System User.
Priority	Nice to have. This functionality is not essential to the operation of the HMM and may be included if time permits.
Preconditions	1. Transition must be Selected. (<i>See Select Single Transition</i>)
Flow of Events	1. User chooses the desired Appearance Setting from amongst the following options. <ol style="list-style-type: none">i. Line Thickness (<i>Sets the thickness of the Outline</i>)ii. Line Colour (<i>Sets the colour of the Outline</i>)
Sub Cases	None.
Extensions	None.

A.25 Create new File

Summary	In order to be able to specify a HMM a new file must be created.
Actors	System User.
Priority	Must have. The ability to create a new file is essential to be able to specify a HMM.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>New</i> Option.2. System prompts the user to specify a filename.3. User enters the desired filename and confirms.4. System creates the file and initialises the Chart to allow the user to draw the HMM.
Sub Cases	None.
Extensions	None.

A.26 Save HMM to File

Summary	The user may wish to save the HMM representation to a file for possible retrieval in the future.
Actors	System User.
Priority	Should have. This functionality is not essential to the operation of the HMM but should be included to save rework.
Preconditions	A File must have been created. (<i>See Create new File</i>)
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Save</i> Option.2. System saves all relevant information about the HMM to the specified file.
Sub Cases	None.
Extensions	None.

A.27 Open HMM from File

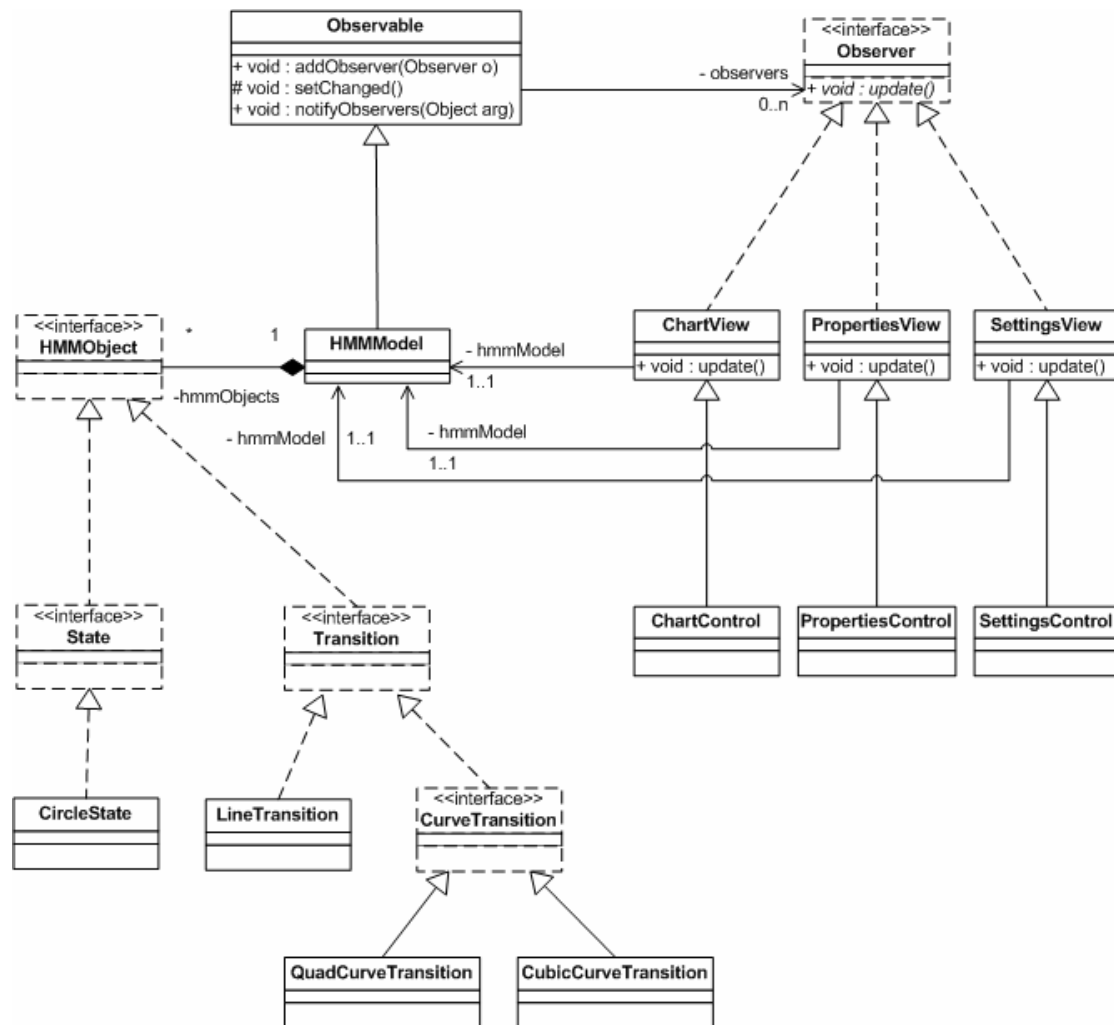
Summary	The user may wish to open an already specified HMM from a saved file.
Actors	System User.
Priority	Should have. This functionality is not essential to the operation of the HMM but should be included to save rework.
Preconditions	None.
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Open</i> Option.2. System prompts user to pick the desired file.3. User chooses the file previously saved by the System.4. System reads the requested file and displays its contents on the screen.
Sub Cases	None.
Extensions	None.

A.28 Generate XML File

Summary	Once the HMM has been specified the user may wish to generate the XML representation.
Actors	System User.
Priority	Must have. This functionality is essential to the achieve the desired XML representation of the HMM.
Preconditions	A File must have been created. (<i>See Create new File</i>)
Flow of Events	<ol style="list-style-type: none">1. User chooses the <i>Generate XML</i> Option.2. System translates the HMM representation in Graphical format into XML format according to the defined rules and constraints.
Sub Cases	None.
Extensions	None.

Appendix B

UML Class Diagram



Appendix C

Casino HMM Saved File

```
<alphabets>
  123456
</alphabets>

<sequences>
  1
</sequences>

<algorithms>
  <forward>
    true
    true
    false
  </forward>
  <backward>
    true
    true
    true
  </backward>
  <viterbi>
    true
  </viterbi>
  <sample>
    false
  </sample>
</algorithms>

<tables>
  <table>
    FAIR
    1/6
    1/6
    1/6
    1/6
    1/6
    1/6
  </table>
  <table>
    LOADED
    1/10
    1/10
    1/10
    1/10
    1/10
    1/2
  </table>
</tables>
```

```
<states>
  <state>
    <label>
      START
    </label>
    <type>
      1
    </type>
    <prob>

    </prob>
    <position>
      300
      300
    </position>
    <shape>
      0
    </shape>
    <appearance>
      1
      1
      -16777216
      -1
      -16777216
      Dialog
      10
      true
      false
    </appearance>
  <state>
  <state>
    <label>
      FAIR
    </label>
    <type>
      0
    </type>
    <prob>

    </prob>
    <position>
      500
      200
    </position>
    <shape>
      0
    </shape>
    <appearance>
      1
      1
      -16777216
      -1
      -16777216
      Dialog
      10
      true
      false
    </appearance>
  <state>
```

```
<state>
  <label>
    LOADED
  </label>
  <type>
    0
  </type>
  <prob>

  </prob>
  <position>
    500
    400
  </position>
  <shape>
    0
  </shape>
  <appearance>
    1
    1
    -16777216
    -1
    -16777216
    Dialog
    10
    true
    false
  </appearance>
<state>
<state>
  <label>
    END
  </label>
  <type>
    2
  </type>
  <prob>

  </prob>
  <position>
    700
    300
  </position>
  <shape>
    0
  </shape>
  <appearance>
    1
    1
    -16777216
    -1
    -16777216
    Dialog
    10
    true
    false
  </appearance>
<state>
</states>
```

```
<transitions>
  <transition>
    <state>
      START
      FAIR
    </state>
    <prob>
      0.5
    </prob>
    <position>
      331
      284
      469
      216
    </position>
    <behaviour>
      true
      true
    </behaviour>
    <shape>
      0
    </shape>
    <appearance>
      1
      -16777216
    </appearance>
  <transition>
  <transition>
    <state>
      FAIR
      FAIR
    </state>
    <prob>
      0.95
    </prob>
    <position>
      510
      166
      490
      166
      553
      129
      447
      129
    </position>
    <behaviour>
      false
      false
    </behaviour>
    <shape>
      2
    </shape>
    <appearance>
      1
      -16777216
    </appearance>
  <transition>
```

```
<transition>
  <state>
    FAIR
    LOADED
  </state>
  <prob>
    0.05
  </prob>
  <position>
    490
    234
    490
    367
    456
    300
  </position>
  <behaviour>
    false
    false
  </behaviour>
  <shape>
    1
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
<transition>
  <state>
    LOADED
    LOADED
  </state>
  <prob>
    0.9
  </prob>
  <position>
    510
    434
    490
    434
    553
    471
    447
    471
  </position>
  <behaviour>
    false
    false
  </behaviour>
  <shape>
    2
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
```

```
<transition>
  <state>
    LOADED
    FAIR
  </state>
  <prob>
    0.1
  </prob>
  <position>
    510
    366
    510
    234
    544
    299
  </position>
  <behaviour>
    false
    false
  </behaviour>
  <shape>
    1
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
<transition>
  <state>
    FAIR
    END
  </state>
  <prob>
    0.5
  </prob>
  <position>
    531
    216
    669
    284
  </position>
  <behaviour>
    true
    true
  </behaviour>
  <shape>
    0
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
```

```
<transition>
  <state>
    LOADED
    END
  </state>
  <prob>
    0.5
  </prob>
  <position>
    531
    384
    669
    316
  </position>
  <behaviour>
    true
    true
  </behaviour>
  <shape>
    0
  </shape>
  <appearance>
    1
    -16777216
  </appearance>
</transition>
<transitions>
```

Bibliography

- [1] Supercomputing Institute, Univeristy of Minnesota. HMMER User Guide.
- [2] Alter Steven 1999. Information Systems: A Management Perspective. Third Edition. *Addison-Wesley Publishers Limited*.
- [3] Pressman Roger 2004. Software Engineering: A Practitioner's Approach. Sixth Edition. *McGraw-Hill Publishers Limited*.
- [4] James Gosling, Henry McGilton 1996. The Java Language Environment. *Sun Microsystems Ltd*.
- [5] Singh Mona, 1999. Topics in Computational Molecular Biology. *EMBnet*.
- [6] Durbin R., Eddy S., Krogh A., Mitchison G. 1998. Biological Sequence Analysis. *Cambridge University Press*.
- [7] Net-ID. <http://www.netid.com>
- [8] eNode. <http://www.enode.com>
- [9] L. R. Rabiner, 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition, *Proceedings of the IEEE, Vol. 77, No. 2*.
- [10] Burge Cristopher, 1997. Identificatio of Genes in Human Genomic DNA. *PhD. Thesis, Stanford University*.
- [11] Howard R. A. 1971. Dynamic Probabilistic Systems Vol. II: Semi-Markov and Decision Processes. *John Wiley & Sons, New York*.
- [12] Russell S., Norvig P. 2003. Artificial Intelligence: A Modern Approach. 2nd Edition, *Prentice Hall*.
- [13] Jurafsky D., Martin J. H.2000. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. *Prentice Hall*.