TELL ME MORE: ADDING NARRATIVE SCHEMA TO THE RUBE SIMULATION FRAMEWORK

By

JOHN HAYS

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003
To my parents,
Fred and Sherry Hays,
And all of my friends,
Without whom none of my success would be possible
ACKNOWLEDGMENTS

I would like to thank my thesis committee chairman, as well as digital arts and sciences graduate advisor, Dr. Paul A. Fishwick, for helping to guide me through my entire journey as a graduate student at the University of Florida. I would also like to thank Dr. Jane Yellowlees Douglas for not only being on my thesis committee, but for constantly inspiring and enlightening me in the area of interactive narrative. I also give much thanks to Dr. Joachim Hammer for his time and interest in being on my thesis committee.

My sincerest gratitude goes out to those who helped me with the technical side of my thesis. These people include Kristian Damkjer and Minho Park. Without their great devotion of time and knowledge, this thesis would not be possible.

I, of course, would like to thank my parents, Fred and Sherry Hays, for all their love and support. I would also like to thank the rest of my extended family for always believing in me. Finally, I would like to thank God for my continued existence, and for everything that has ever or will ever happen to me.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Challenges</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Organization of the Thesis</td>
<td>1</td>
</tr>
<tr>
<td>2 BACKGROUND</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Numeric Abstraction</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Aesthetic Computing</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Narrative Schemas</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Creating Narrative Structure</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Conclusion</td>
<td>12</td>
</tr>
<tr>
<td>3 RELATED WORKS</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Multitalk</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Random Text Generators</td>
<td>13</td>
</tr>
<tr>
<td>3.3 Papert Constructionism</td>
<td>14</td>
</tr>
<tr>
<td>3.4 Protégé</td>
<td>15</td>
</tr>
<tr>
<td>4 THE PROCESS</td>
<td>16</td>
</tr>
<tr>
<td>4.1 Initial Concerns</td>
<td>16</td>
</tr>
<tr>
<td>4.2 Designing the Schema</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Designing the Translator</td>
<td>18</td>
</tr>
<tr>
<td>4.4 Creating a World with Narrative Schema</td>
<td>19</td>
</tr>
<tr>
<td>5 CONCLUSION</td>
<td>22</td>
</tr>
<tr>
<td>5.1 Results</td>
<td>22</td>
</tr>
<tr>
<td>5.1.1 Example World</td>
<td>22</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1. SSML Schema Code.</td>
<td>17</td>
</tr>
<tr>
<td>4-2. NFE XSLT Code.</td>
<td>18</td>
</tr>
<tr>
<td>4-3. Simple SSML File.</td>
<td>19</td>
</tr>
<tr>
<td>4-4. Rube Coding Structure.</td>
<td>21</td>
</tr>
<tr>
<td>5-1. Complex Example World.</td>
<td>22</td>
</tr>
<tr>
<td>5-2. Simple FSM.</td>
<td>23</td>
</tr>
<tr>
<td>5-3. Complex Nested FSM.</td>
<td>24</td>
</tr>
</tbody>
</table>
Aesthetic computing is a term coined by Fishwick, at the University of Florida. It is used to describe the idea of visualizing mathematical models in order that they might be more easily understood by people not already familiar with the concepts.

Stories provide another way for people to understand and share ideas in a way that is both universally understood and as old as time itself. Narrative schema provides a way to analyze and provide structure for these stories. This allows stories to be defined in a scientific and ordered way.

In my research, I am attempting to map narrative schema within an aesthetic computing application environment called rube. Rube uses XML, VRML, X3D, and other languages to allow users to create a visual world with models and animations, and then add behaviors to that world which cause it to act like the mathematical model they want to visualize. My focus will be on using the structured format of narrative schema to add a new dimension to the rube format.
I have picked one particular narrative schema which is simple and straightforward. It consists of three basic types of states: initial steady state, conflict state, and redress state. These states can then be subdivided into lower levels, allowing for subplots, character motivations, and so forth.

My ultimate goal is to allow users to input the basic elements of the narrative schema for whatever particular story fits state for state onto the mathematical model they happen to be simulating. This will use my own XML story schema language, SSML, which will organize the information, combine it with X3D, and feed it into the rube architecture.
CHAPTER 1
INTRODUCTION

1.1 Challenges

The goal of this research is to create a mechanism that will map story structure onto formal models so that these formal models are more easily understood, remembered, and appreciated by those not familiar with formal models. The abstract idea is to think of narrative as an additional dimension, like 3D, audio, or touch, and then add it to the virtual visualization of a mathematical structure, in this case a finite state machine (FSM).

The key difference with this research and, say, simply mapping a particular story onto a particular FSM is the fact that a particular story has a fixed structure and therefore will only work with the FSM that fits that structure. This does not allow for extensibility or modularity. What this research accomplishes is to create an actual story schema from which any possible narrative can be created and then mapped onto virtually any FSM.

This research will strive to accomplish the following tasks: find or create a narrative schema that is well formed, yet flexible enough to conform to any FSM, and figure out a way to successfully map the resulting stories onto FSM’s in the rube architecture.

1.2 Organization of the Thesis

Historical background on all aspects from aesthetic computing to narrative schema is discussed in Chapter 2. Chapter 3 is a brief survey of other recent attempts to connect narrative schema and computer science technologies. Chapter 4 describes the process
involved in creating the story schema modeling language (SSML), as well as how it currently connects to rube. The schema and the extensible stylesheet language transform (XSLT) file are both discussed, along with how to create a world that includes narrative mapping with SSML. Chapter 5 contains the conclusion to the research, discussing results as well as future possible work.
CHAPTER 2
BACKGROUND

2.1 Numeric Abstraction

How do people represent the basic necessities of life? Looking back to early times, it is seen to be very personal, very individual. A person growing corn on a farm might want some meat for dinner, so he would attempt to trade with a man who raised cattle. This, however, proved unwieldy when it came time to discuss terms of a trade. One could not just drag cows, corn, or whatever else each person identified with everywhere he went. It was not financially feasible or physically practical.

Thus, an abstraction was made. Cows and corn became clay tokens. This was still very personal, since the tokens still represented real items. Every different token shape was unique to the object it represented. Sheep, jars of oil, and other things were all represented with their own tokens [14].

Sumerian tokens actually abstracted even more in that they stood for units of merchandise, meaning that each counter stood for measures of items, instead of each item individually [13]. This still was not a purely abstract numeric system, since the tokens only stood for amounts of their respective items, not amounts of any item. This idea of measurement was, however, the impetus for pure abstraction. Terms such as “decade” and “hundred” derived from measurements such as inch, foot, and yard [8]. This idea led to an actual number system, which was eventually expressed not as tokens but as ink on paper, chalk on slate. Numbers were then combined with symbols to form expressions that represented more complex ideas. By this time it was all very abstract and devoid of
personal meaning, yet this was the way it was done to conform to the economic and physical limitations of the time. It is perfectly understandable that this needed to be done, but the fact that this same way of doing things has carried on up to the present day is, when analyzed closely, quite shocking.

Today, even in high-level universities, the most complex of ideas are represented with nothing better than circles and squares drawn with chalk on slate. In an advanced computer architecture classroom, the inner workings of a personal computer are represented by two rectangles drawn on the board, one for the central processing unit and one for the arithmetical logic unit. In a simulation class, mathematical systems such as finite state machines and entity relationship diagrams are represented with circles connected by lines, once again drawn on the board. With all the technological devices available to us that enable us to visualize things in new and more immersive and engaging ways, one would think something could be done to change the way we learn, giving us back the personalization we once had.

2.2 Aesthetic Computing

In fact, attempts are being made to change all of that. One attempt, in particular, is a project called rube. Rube deals with aesthetic computing, a term coined by Fishwick at the University of Florida. Aesthetic computing is the attempt to visualize mathematical and computing concepts in order to provide a better understanding to people not educated in the area, as well as to recapture the personalization of data representation lost to us since ancient times. Fishwick’s main thrust is the simple idea that, if given any physical or technological material required, as well as an extremely fast and efficient construction process so as to make the final product physically and financially feasible, how would we
then choose to represent everyday concepts currently bound to chalk on slate or pencil on paper?

The motivation behind why anyone would bother to change the way one represents mathematical data can be thought of in the way a child learns. In the very early stages of learning, an adult presents a child with three dimensional objects, such as balls and blocks. The child can touch and feel these objects, and has instant recognition of their existence and their inherent importance to the child. In this way the child learns concepts such as addition and subtraction that otherwise would be extremely difficult to get across in a purely abstract manner. By that same token, aesthetic computing is the hope that one day a child, or any person not specifically educated in the field of mathematics, can learn much more difficult and technical mathematical concepts in ways that they can instantly feel comfortable with, such as three dimensional objects, or in the case of this research, a story based on a structured and familiar schema.

The advantage of using narrative as opposed to simply using three dimensional objects is the fact that it would play to a different audience, or at least a different way for the same audience to understand. It is well known that different people have different ways of learning, and in fact most people can learn things in multiple ways. It is also obvious from the popularity of films and books that people have a fondness for stories, both for education as well as entertainment. People spend massive amounts of money on DVD’s, video games, books, and so forth, which all basically start out as stories on paper and go from there. It is for this reason that adding narrative seems like the next logical step in helping someone not educated in mathematics to understand mathematical models represented visually. It is the hope that when someone is presented with a mathematical
model that has a story mapped onto it, the person can then read the story as the step through the model, and understand the structure of the model through the structure of the story.

The ability to create in this manner will push the idea of aesthetics into the forefront of model representation, with artists taking the lead on design. This idea of "aesthetic computing" will bring art and science together, with each in direct support of the other [5]. The rube project is the implementation of this idea using XML and VRML technologies to create virtual worlds that house these models. The vision of the rube project is to use as many different ways as possible to visualize data, thereby opening it up to the largest possible audience. Different genres, such as architecture, theatre, and art, have been used to visualize these models for the broadest possible audience. So far, architecture has been one of the most successful approaches used. Mathematical structures have been visualized as buildings or objects in order to make them palatable to people familiar with those various structures.

2.3 Narrative Schemas

A completely different approach being attempted in this thesis is to use narrative structures. That is, to map a mathematical or computing structure onto a story structure, thereby providing an understanding of the mathematical structure to people familiar with whatever story might be chosen or created. In order to have a better understanding of how to map a narrative structure onto a mathematical structure, it is necessary to understand some of the history surrounding the current state of narrative structures.

Schank did some early work on narrative structures, or schemas, for the area of artificial intelligence. His main concern was to understand how people processed information from other people in such a way as to correctly understand the underlying
meaning of the speaker. Schank decided that people had two types of knowledge, general and specific. General knowledge allows people to understand the basic actions of others, whereas specific knowledge allows people to interpret and participate in events that are repetitious. By forming mental structures, or schemas, of these events in our minds, we are able to reenact them at any time. For example, if you read a snippet from a larger story that simply says, “John went to a restaurant. He ordered some food. He paid the check and left.” You will assume, through your own personal schema regarding restaurants, that John actually received the food that he ordered, even though it is never specified in the text.

Schank had a very specific view of how people understood each other. “Understanding is knowledge-based. The actions of others make sense only insofar as they are part of a stored pattern of actions that have been previously experienced” [12:17]. This means that the ways in which a person understands the actions of another are based on any similar actions that person might have encountered before. Schank thought that these actions formed patterns, which he called scripts.

Schank then described a second mechanism needed for understanding, a script applier, as something that fills in the gaps left out of a story by referencing a script relevant to the story. In other words, as someone is listening to a story, they are forming a causal chain of events. Once there is a break in this chain in the mind of the listener, they must apply one of their own scripts to the story to attempt to fill in the gap and continue the chain.

Now came the crucial part that gave meaning to scripts from the story telling point of view:
When someone decides to tell a story that references a script, he recognizes that he need not . . . mention every detail of his story. He can safely assume that his listener is familiar with the referenced script and will understand the story as long as certain crucial items are mentioned [12:32].

As mentioned before when explaining the idea of schemas, this means that stories can leave out trivial details that the author assumes the reader is already familiar with. This allows a narrative mapping to fit onto a structure without having to worry about mapping unnecessary details.

Schank does a nice job summarizing the main point of using scripts in storytelling in the following manner. “Thus, while it is possible to understand a story without using a script, scripts are an important part of story understanding. What they do is let you leave out the boring details when you are talking or writing, and fill them in when you are listening or reading” [12:35]. This is a crucial step in the process of applying narrative schema to a formal model. As mentioned above, this allows the unneeded details to be left out of the mapping, leaving only those parts applicable to the model itself.

While this does a good job forming a basis as to how a person can understand a narrative that is based upon some type of schema, or script in Schank’s case, it does not provide a nice high level schema with which to form these narratives. Narratives, as Douglas puts it, must obey some type of order for us to use personal scripts to understand them. “Scripts can help nudge us toward comprehension at a local level only if we can recognize individual elements as part of a larger script . . . what enables us to understand what we see or read, is the relative fixity of the larger script” [4:3].

This is to say, people can understand schemas on their own, separate from larger narrative structures, but in order to form any type of lengthy narrative, you must create some high level narrative schema for the entire piece. In this way, you can then call upon
that overall narrative schema for not only the main narrative plot, but also for any smaller subplots you wish to add, adding depth and interest to your story.

2.4 Creating Narrative Structure

The next major step in a good structure for creating a narrative schema comes about from Beaugrande and Colby’s definition of an “interesting narrative” as “evoking a world in which the action or event which changes the initial state to the goal state is not so obvious that it would probably happen anyway in the normal course of things…there must be substantial uncertainty about state-event-state chaining if the audience is to participate actively” [2:3]. This is where we start to break away from Flanagan and Allen’s vague “beginning, middle, end” definition. This definition now gears narrative toward an active audience, which is ideal for our intentions with the rube project, where the very nature of the environment is geared toward interactivity.

Beaugrande and Colby go on to provide the two basic types of characters that should be included in a narrative schema, the protagonist and antagonist. “When the various events and states are assigned values according to one character’s plans and goals, that character will be the protagonist if the audience shares those values, and the antagonist if the audience rejects them” [2:4]. This emphasis on the perspective of the viewer shows the personal nature of viewing such a narrative, which is exactly what we are going for…a move away from the abstract, back to the personal. The actions of these characters are also nicely defined as a “mutual influencing and blocking of event-state pathways” [2:4]. In this way, we get the idea of choices, or forking pathways, as in an FSM structure.

Beaugrande and Colby actually formulate a possible schema out of these ideas in the following manner:
• Identify character
• Create problem state for that character
• Identify a goal state for that character
• Initiate a pathway from the problem state to the goal state
• Block or postpone attainment of the goal state
• Mark one state transition as a turning point
• Create a terminal state that is clearly marked as matching or not matching the goal state.

While this is a very nice model for a possible narrative schema, it is still very linear, especially with the terminal state. However, Kintsch immediately deals with this: “This rule sequence is of course recursive, so that it may generate either a minimal story or an episode in a longer story” [9:3].

The two-character, protagonist and antagonist, version is quite similar, with a few allowances made for the second character. That is, two characters are introduced instead of one; the problem state for the first character is now caused by the second character; each character’s goal state is undesirable for the other character; each character’s actions block the progress of the other character, either somewhat or totally; the turning point state is decisively enabling for one character, disabling for the other; the terminal state sees a goal attainment for one of the characters. The seeming weakness in this case would be the existence of only two characters. What if three or more were needed for the mapping? A modified rule is supplied for that case, where one of the characters is given comrades in an effort to better block the other character. Each of these comrades can then be treated as a character in their own right. The problem there is that each character would then seemingly need the entire set of rules applied to them. Beaugrande and Colby
afford the reader a bit of leeway in this area: “The realization of all rules for every story or episode is not obligatory…if a given rule is not explicitly realized; an audience will infer a default realization” [2:4]. This harkens back to Schank’s scripts, where the reader fills in any gaps with their own expectations of the narrative schema.

Beaugrande and Colby further the parallel between narrative and FSM’s when they quote Rumelhart’s definition of a narrated world as a “progressive system in which events and actions occur and bring about changes from one state to the next” [2:3]. They use Wright’s definition of actions as intentional human acts that change the state of affairs. Since finite state machines consist of states and transitions, this is a very close parallel, and should map well. Dijk reinforces the idea of sublevels by saying that discourse is composed of a subclass of actions [9]. Therefore, multiple subclasses of actions, leading from state to state, could combine into a rich narrative structure.

The key idea that Beaugrande and Colby put forth in their argument, aside from their highly useful schema structure, is the combination of ideas by Schank and Rumelhart of “well-formedness” in a narrative schema structure. “Well-formedness of stories depends…upon the mutual accessibility of states and actions, and upon the relevance of actions to goal attainment. The audience expects that states and actions will not be tied into sequence unless each pair is linked by a discoverable relationship of cause, reason, enablement, or purpose” [2:4]. This continues to advance the requirements of a proper narrative schema structure from the basic “beginning, middle, end” that we started out with. It also continues the early ideas of Schank, requiring the readers to infer what those reasons and purposes might be if not given explicitly in the story, as stated above.
Bruner furthers the idea that narrative needs some type of causation by saying that narrative deals with the vicissitudes of human intentions [3]. He even goes so far as to say that people see causality in everything around them. Bruner then provides a very nice, simple skeleton on which to base our narrative schema structure for mapping purposes. “Life-like narratives start with a canonical or “legitimate” steady state, which is breached, resulting in a crisis, which is terminated by a redress, with recurrence of the cycle an open possibility” [3:4]. He also deals with the ever-present problem of linearity. “The story stuff, or fabula…could be plotted in linear order, in flashbacks, or even in medias res, starting virtually anywhere” [3:4].

2.5 Conclusion

Thus, by combining Bruner’s schema skeleton with Beaugrande and Colby’s basic rules, we can form a well-structured narrative schema useful for our mapping purposes. We should of course continue to keep in mind the underlying reasons as to how and why people are able to understand this schema, as explained by Schank and others. This will help us to avoid problems arising from careless mapping procedures that might confuse and alienate the user.

Once a successful narrative schema can be created, a mapping structure will then be coded to map the schema onto the rube architecture, allowing the user to map any narrative onto a mathematical structure such as an FSM. This will then allow a person unfamiliar with FSM’s to have a better understanding of the concept through a familiar narrative structure. If this idea gains popularity, it could ultimately change the way in which people learn about many complex ideas, allowing for a re-personalization not seen for quite some time.
CHAPTER 3
RELATED WORKS

3.1 Multitalk

Multitalk [6] is a schema for creating combinatorically flexible stories. The research was motivated by Murray’s philosophy on interactive narrative, and was written for her course of the same name. The Multitalk language is object oriented, and is composed of objects that represent the various things in the world, and morphemes that represent the actions that are applied to the objects. The language uses morphemes in order to be modular, instead of the usual tree based approach for interactive programs. Morphemes require specific preconditions to occur before they activate, allowing for multiple results with each interaction.

The test case given on the website uses a grammar similar to the SSML schema. It is comprised of the following: possibility, frustration, resolution, and conclusion. The difference is that this language is created purely for story interaction. It does not attempt any type of mapping onto mathematical models as SSML does.

3.2 Random Text Generators

Random text generators, such as the Dada Engine, RMutt, and others use recursion to randomly generate text files from basic grammars. The problem with each of these is that, for one, the grammar is never rich enough to come close to replicating a real language, especially the English language, and two, a randomly generated piece by its
nature would have trouble achieving any narrative cohesion, and would do little to help further a user’s understanding of mathematical models.

3.3 Papert Constructionism

Papert worked in the area of learning processes, specifically for children. He viewed the current way of teaching mathematics to be analogous to the teaching of a “dead” language. “I maintain that construction of school math is strongly influenced by what seemed to be teachable when math was taught as a “dead” subject, using the primitive, passive technologies of sticks and sand, chalk and blackboard, pencil and paper” [11:21]. Papert believed that there was a better, more natural way to learn mathematics. He believed that children can learn complicated mathematical concepts through simplistic means that they are familiar with at an early age. “If we really look at the “child as builder” we are on our way to an answer. All builders need materials to build with…In the case of formal mathematics, there is both a shortage of formal materials and a cultural block as well” [11:32]. Thus, Papert believed that there was a need for some type of personalized set of “building blocks” for children to work with in order to learn complex mathematics, just as they had used actual blocks to learn basic mathematical concepts.

Papert used the idea of metaphor to create a program that used a “turtle”, which walked around and drew lines in whatever manner was coded by the user. “The idea of programming is introduced through the metaphor of teaching the Turtle a new word” [11:36]. In this way, children learned very basic programming rules through the direct visual responses of the turtle.
3.4 Protégé

Protégé is a piece of software used for the creation and manipulation of ontologies. Ontologies contain knowledge about a specific domain in a structured manner. They can have many uses, such as knowledge sharing and recycling. In a technical or programming sense, ontologies can be thought of as specifications for specific ideas. Protégé is a way of creating ontologies using programming methodologies. Protégé also can include visualizations such as tables, diagrams, and animations to display data. It also acts as a library for access to other collections of ontologies.

While looking at Protégé, I found it to be extremely useful for ordering ontological data into an object oriented structure similar to the Java programming language. It ended up, however, being more complex than what I was looking for. I did not require the type of organization that it provided. I was looking for more of a simplistic interface for the user, which lets them write a story with as little superfluous requirements as possible.
4.1 Initial Concerns

The biggest technical issue concerning the use of narrative schema for the rube project is that it services the mathematical models, not the other way around. In other words, it would be easy enough to say, given a certain story, create an FSM for that particular story. This, however, is not what we are looking for here.

What we want is the ability to take an existing FSM, and then create a narrative that maps directly onto that FSM. To do this, we need specific mapping rules for the narrative schema that allows it to be flexible enough to accommodate the creation of a story for any FSM.

4.2 Designing the Schema

To implement the schema I had researched, I developed a language that I termed Story Schema Modeling Language, or SSML. This language was created as an XML language, with an XML schema definition. Any user wishing to use the narrative schema must create an XML file that obeys the SSML coding rules.

In order to maintain the particular story schema I have chosen to work with, I was forced to create a minor limitation dealing with the creations of FSM’s. The narrative schema requires the following at the simplest level: steady state, conflict, redress, and an optional final steady state. Thus, any FSM that uses this particular story schema must be composed of at least three states.
Figure 4-1. SSML Schema Code.

Just as in any XML language, any file written in SSML (Story Schema Modeling Language) starts and ends with the <SSML> opening and closing tags. In the schema, this tag is made up of an SSMLType complex type, which references a “states” group. The states group is then made up of the sequence of states required for SSML. This includes a mandatory steady state, written as <steady> when coding a sample file, followed by a group named “conred” in the schema. The “conred” group is coded in that way because conflict must always be followed by redress, so grouping the two was a coding optimization. This forces the user to then code <conflict> and <redress> states at least once following the <steady> state.

Finally, there is another reference to the steady state, this time optional, since there does not necessarily need to be a final steady state. Each of the states in the schema then reference an element called “script,” which is the tag used before and after all actual string text entered into any SSML file for the story. Since a film script is where the words are stored, <script> seemed like a logical choice for a naming convention here.
4.3 Designing the Translator

One major aspect of figuring out the NFE (Narrative Fusion Engine) XSLT for the SSML language was maintaining a one-to-one relationship between the VRML file’s text nodes and the SSML file’s script nodes. Originally, all script nodes were going into one text node. That obviously was incorrect, so a variable had to be created to keep track of the position of each text node and then match them up with their corresponding script nodes.

![Figure 4-2. NFE XSLT Code.](image)

Another challenge was displaying the narrative output in a readable manner in VRML. Text that is simply dumped between two sets of quotation marks in the text part of a billboard node in VRML is output all on one continuous line. This obviously would not work with the idea of reading a story, since most stories wrap continuously to form a page of text.

The next step was to figure out how to wrap in VRML. As it turns out, VRML wraps by adding a closing set of quotes and a comma, followed by another opening set of quotes. So, in the XSLT file, a check should ultimately be added to find every occurrence of either a certain number of characters or an ASCII enter if the user prefers,
and add those three ASCII characters at the end of each occurrence. At the present time, the XSLT does not yet include this addition, so the user must add the code to the VRML file directly.

4.4 Creating a World with Narrative Schema

To create a world using the SSML narrative schema, the first step is to take the FSM that you are applying the mapping to, and either choose an existing story that maps correctly to the FSM, or create a story using the narrative schema rules of SSML. Once a narrative is chosen, the SSML file should be created. Due to the literal translations of rube, all text in the SSML file must start and end on the same line. In other words, a single line of actual story text should start and end with the <script> tag. Also, apostrophes are not correctly translated by rube, and therefore are not allowed.

```
<SSML>
  <steady>
    <script>Adam and Eve frolic in Eden.</script>
  </steady>
  <conflict>
    <script>Adam and Eve bite the apple and are cast out of Eden.</script>
  </conflict>
  <redress>
    <script>Adam and Eve apologize profusely and are allowed back into Eden.</script>
  </redress>
</SSML>
```

Figure 4-3. Simple SSML File.

The next step is to create the VRML geometry that will be the architecture for the narrative mapping. This includes all text nodes for each narrative state. The text nodes should not include the string attribute, since the NFE XSLT will create this. The VRML world must then be translated to X3D in order for the NFE XSLT to function properly.
Once the world is in X3D form, the SSML file and the X3D file are combined using the NFE XSLT. The X3D file is then translated back into VRML. Until a more advanced XSLT is created, the next step is to add “,” to each line of text so that it wraps correctly in the resulting VRML file. At this point, the world is ready for rube implementation.

The next step is to create an MXL (Multimodel eXchange Language) file for the mathematical model structure. Using rube in its current incarnation requires an FBM to act as the root node, but soon that will not be a requirement. After the MXL file is created, the javascript must be created to add functionality to the world. Once all files are created, they may then be merged using rube software. The final step is to edit the resulting world, adding proper variable types, javascript code, and routing.
Figure 4-4. Rube Coding Structure.
CHAPTER 5
CONCLUSION

5.1 Results

5.1.1 Example World

The world I have created as an example uses the version of rube current at the time of my coding. This version does not allow nested FSM’s, so I rooted my world in an FBM, with an FSM nested inside of it. With the current work being done on rube implementation, the updated MXL schema will support nested FSM’s.

Figure 5-1. Complex Example World.

The FBM consists of three blocks: an input block, a block containing an FSM, and an output block. The input block uses a simple random number generator to provide input to the FSM. The current state number is shown visually in the first block. The
FSM was originally designed as a multimodel FSM to show how narrative schema would work in a multimodel, but has been coded as one large FSM to conform to the current version of rube as of this writing. The text wall that represents the current state changes color from blue to red in the second block. The third block displays the narrative text at the current state. Users may click through the viewpoints to familiarize themselves with the straightforward narrative structure in the FSM, and thus have a better understanding of what they see in the third output block of the FBM.

The fact that spheres and cylinder are used for the FBM might seem strange when the entire idea of rube is to get away from abstract representation, but my aesthetic focus was on the FSM representation, as opposed to the FBM representation. My original plan was to show the FSM as a complex model, but the current version of rube does not support nested FSM’s, so I was forced to use the FBM as the larger model. Since an extremely similar FBM structure had already been implemented by another graduate student, I simply used the same structure. The fact that it consisted of spheres and cylinders actually ended up being mere coincidence, and does not reflect any particular aesthetic design. It is used merely as a simple shell for my FSM model.

Figure 5-2. Simple FSM.
Figure 5-2 provides an example of a simple linear FSM with narrative mapped to each state, or VRML wall. This is how my example had to be coded using the current version of rube. Figure 5-3 shows my original intention of implementing my example as a complex nested FSM, with simple FSM’s nested inside the larger complex one.

5.1.2 Other Possible Implementations

Since the SSML narrative mapping is used as an aesthetic addition to the scene graph, many different implementations are possible. One idea could be to map an FSM onto an architectural model, such as a house, and then map the narrative onto the house. In this idea, each room could be a state, with halls acting as edges, connecting the rooms. In a multimodel system, the rooms could be nested models, with the halls being the simple states between the complex ones.

5.1.3 Concrete FSM’s

For FSM’s that model real-world phenomena or objects, the narrative could then reflect the target object or phenomena. The writer would have to create a story that
corresponded correctly with each state of the FSM, as well as the genre of the target object. For instance, an FSM of a light bulb could have a narrative mapping that told a story involving a light bulb in some way, with each state of the narrative having some meaning that corresponds to the current state of the light bulb. Another alternative could be to have the current narrative state become readable by the action of the corresponding state of the light bulb. For instance, when the light bulb turns on, the text for that state becomes readable, and when the light bulb turns off, the text for the on state disappears, and the text for the off state becomes readable.

5.1.4 Branching FSM’s

For FSM’s that are conditional, or branch for whatever reason, corresponding narratives can be created for each condition. Stories like “Choose Your Own Adventure” that have branching possibilities would be ideal for FSM’s that branch to unique finite ends. For FSM’s that branch back into themselves, stories that revolve around fatalistic views could work.

In other words, these would be stories that operate on the idea that, no matter which path you take in life, the end result is the same. An example of this would be the recent cinematic remake of Wells’ The Time Machine, where the main character goes back in time over and over in order to prevent the death of his wife, only to realize that she will die no matter what he does.

5.2 Future Work

5.2.1 Avatars

In order to attain a higher level of personalization and realism in the VRML visualization of rube, avatars could be used to act out stories. Avatars actually have dual definitions. In VRML, an avatar is a symbolic-virtual-world representation of a real-
world person [1]. Another definition for avatars is that they are an archetype. This plays into the idea of narrative schema perfectly, in that using archetypes keeps the stories open to the widest audience possible.

Efforts are currently underway to add Blender support to the rube architecture. Blender is a 3D package, similar to 3D Max and Maya, which allows the user to create 3D models and animation in a visual manner, as opposed to the coding interface of VRML. Ultimately, users will be able to visually create a world that is visually and artistically rich, and then export it in a manner that runs it through the rube architecture, creating a VRML world that includes simulation animation. Using this interface, users could then easily create avatars for the narrative mappings onto their FSM’s.

5.2.2 Further Integration into Rube

A different use of the Blender interface could allow the narrative to be included visually. Blender supports the use of text objects, similar to 3D Max and Maya. Users could conceivably type their stories directly into the Blender interface, and then wrap them in whatever manner they wish, placing them wherever they want. This would eliminate the need for the XSLT file completely.

5.2.3 Narrative in Game Design

The idea of narrative schema mapping onto mathematical models, such as FSM’s, could prove to be revolutionary in the area of game design. It could be the first step in a true marriage of writers and engineers on a game design team. For example, if writers come up with a compelling story, they could then easily put it into an SSML format, allowing the engineers to see the technical structure of the story so that they can then be aware of how they need to the design the game levels as well as how many decisions
must be made by the AI in the game. Similarly, if the engineers design the levels and
code of the gaming world, the writers can then write a narrative to match that world.

As far as online multiplayer games, users could easily create their own “stories” to
correspond with the design of the world in which they are playing. This would follow the
classic idea of RPG “Dungeon Masters.”
APPENDIX A
SSML SCHEMA

<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:annotation>
  <xsd:documentation xml:lang="en">
    Story Schema Markup Language (SSML) schema for rube architecture.
    Copyright 2003 University of Florida. All rights reserved.
    Coded by John Hays.
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="SSML">
  <xsd:annotation>
    <xsd:documentation>
      This is the SSML root node.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType name="SSMLType">
    <xsd:sequence>
      <xsd:group ref="states" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="script">
  <xsd:annotation>
    <xsd:documentation>
      This is the actual story text.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

<xsd:group name="states">
  <xsd:annotation>
    <xsd:documentation>
      This is the parent group of all states.
    </xsd:documentation>
  </xsd:annotation>
</xsd:group>

</xsd:schema>
<xsd:element name="steady">
  <xsd:annotation>
    <xsd:documentation>This is the steady state, which bookends all other states.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="script" minOccurs="1" maxOccurs="1"/>
      <xsd:group ref="states" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="conflict">
  <xsd:annotation>
    <xsd:documentation>This is the conflict state.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="script"/>
      <xsd:group ref="states" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="redress">
  <xsd:annotation>
    <xsd:documentation>This is the redress state.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="script"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:group ref="states" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:group name="conred">
<xsd:annotation>
    <xsd:documentation>
        This is the group of conflict and redress states.
        Every conflict state must be followed by a redress state.
    </xsd:documentation>
</xsd:annotation>
<xsd:sequence>
    <xsd:element ref="conflict" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="redress" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:group>
</xsd:schema>
APPENDIX B
NFE XSLT

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:param name="SSML"/>
  <xsl:template match="/">
    <xsl:apply-templates select="*" mode="body"/>
  </xsl:template>
  <xsl:template match="@*|node()" mode="body">
    <xsl:variable name="temp" select="name()"/>
    <xsl:if test="$temp = 'Text'">
      <Text>
        <xsl:call-template name="import"/>
        <xsl:apply-templates mode="body"/>
      </Text>
    </xsl:if>
    <xsl:if test="$temp != 'Text'">
      <xsl:copy>
        <xsl:apply-templates select="@*" mode="body"/>
        <xsl:apply-templates mode="body"/>
      </xsl:copy>
    </xsl:if>
  </xsl:template>
  <xsl:template name="import">
    <xsl:variable name="q" select="count(ancestor::Transform/preceding::Text)"/>
    <xsl:variable name="SSML_file" select="document(concat($SSML,'.xml'))"/>
    <xsl:for-each select="$SSML_file//script">
      <xsl:variable name="a" select="position()"/>
      <xsl:if test=" $a = $q + 1">
        <xsl:attribute name="string">
          <xsl:value-of select="." />
        </xsl:attribute>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
C.1 Simplest SSML Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<SSML>
  <steady>
    <script>Adam and Eve frolic in Eden.</script>
  </steady>
  <conflict>
    <script>Adam and Eve bite the apple and are cast out of Eden.</script>
  </conflict>
  <redress>
    <script>Adam and Eve apologize profusely and are allowed back into Eden.</script>
  </redress>
</SSML>
```

C.2 Multiple Conflict/Redress Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<SSML>
  <steady>
    <script>Adam and Eve frolic in Eden.</script>
  </steady>
  <conflict>
    <script>Adam and Eve bite the apple and are cast out of Eden.</script>
  </conflict>
  <redress>
    <script>Adam and Eve apologize profusely, but to no avail.</script>
  </redress>
  <conflict>
    <script>God causes Eve to feel great pain in bearing children.</script>
  </conflict>
  <redress>
    <script>Adam and Eve work hard to make a home and a family.</script>
  </redress>
  <steady>
    <script>Adam and Eve have two sons and are a happy family.</script>
  </steady>
</SSML>
```
C.3 Complex SSML Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<SSML>
<steady>
  <script>Frodo Baggins lives happily in the Shire with his uncle Bilbo Baggins.</script>
</steady>
<conflict>
  <script>Gandalf arrives in the Shire.</script>
</conflict>
<redress>
  <script>Gandalf sees Bilbo use the ring and wonders if it is the ring of power.</script>
</redress>
<steady>
  <script>Gandalf reads about the history of the ring, finds that fire will reveal if it is the ring of power, and throws the ring into the fire.</script>
</steady>
<conflict>
  <script>Frodo must get away from the Shire.</script>
</conflict>
<steady>
  <script>Frodo begins his journey to Brie. Sam, Meri, and Pippin join him.</script>
</steady>
<conflict>
  <script>The hobbits encounter a ring wraith.</script>
</conflict>
<redress>
  <script>The hobbits hide and run, evading the ring wraith.</script>
</redress>
<steady>
  <script>The hobbits make it safely to the Inn of the Prancing Pony in Brie.</script>
</steady>
<conflict>
  <script>Frodo encounters a dangerous-looking man known as Strider.</script>
</conflict>
<redress>
  <script>The hobbits make it safely away from the Shire, but arrive in Brie to find that Gandalf is not there.</script>
</redress>
</SSML>
```
Strider turns out to be a friend and leads the hobbits away from Brie to Rivendell.

Strider and the hobbits journey to Rivendell.

The group is attacked by ring wraiths while cooking dinner on the journey.

Strider fights the wraiths off, but Frodo is stabbed by one of the ring wraiths, poisoning him. Strider and the group pick up an injured Frodo and continue the journey to Rivendell.

Strider fights the wraiths off, but Frodo is stabbed by one of the ring wraiths, poisoning him. Strider and the group pick up an injured Frodo and continue the journey to Rivendell.

The condition of Frodo gets much more serious and the group must stop.

Arwen, an elf, comes to take Frodo to Rivendell faster than the group can.

Arwen takes Frodo to Rivendell. The group makes it to Rivendell shortly after.

Frodo finds out that the ring must be taken to Mordor.

A fellowship is created, and they journey to Mordor. Along the way, the fellowship is broken, but Frodo continues.

Frodo gets to a place where Mordor is visible in the distance.

C.4 Complex Example MXL Code

```xml
<?xml version="1.0" encoding="utf-8"?>
<MXL>
  <!-- Multi-Model Setup -->
  <!-- FBM -->
  <model type="FBM">
    <topology type="GRAPH">
      <script>Strider turns out to be a friend and leads the hobbits away from Brie to Rivendell.</script>
      <steady>Strider and the hobbits journey to Rivendell.</steady>
      <conflict>The group is attacked by ring wraiths while cooking dinner on the journey.</conflict>
      <redress>Strider fights the wraiths off, but Frodo is stabbed by one of the ring wraiths, poisoning him. Strider and the group pick up an injured Frodo and continue the journey to Rivendell.</redress>
      <conflict>The condition of Frodo gets much more serious and the group must stop.</conflict>
      <redress>Arwen, an elf, comes to take Frodo to Rivendell faster than the group can.</redress>
      <steady>Arwen takes Frodo to Rivendell. The group makes it to Rivendell shortly after.</steady>
      <conflict>Frodo finds out that the ring must be taken to Mordor.</conflict>
      <redress>A fellowship is created, and they journey to Mordor. Along the way, the fellowship is broken, but Frodo continues.</redress>
      <steady>Frodo gets to a place where Mordor is visible in the distance.</steady>
    </topology>
  </model>
</MXL>
```
<!-- Set up the blocks -->

<!-- Functional Block 1 -->
<node id="F1" type="FUNCTION">
  <script id="narr.js" func="gen"/>
</node>

<!-- Functional Block 2 -->
<node id="F2" type="FUNCTION">

<!-- FSM -->
  <model id="F2FSM" type="FSM">
    <topology type="GRAPH">

<!-- set up the states -->

<!-- State "Main_InitialSteady_1" -->
  <node id="Main_InitialSteady_1" type="STATE" data_type="INTEGER" start="TRUE">
    <script id="narr.js" func="state1"/>
  </node>

<!-- State "Inner_InitialSteady_1" -->
  <node id="Inner_InitialSteady_1" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state2"/>
  </node>

<!-- State "Inner_Conflict_1" -->
  <node id="Inner_Conflict_1" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state3"/>
  </node>

<!-- State "Inner_Redress_1" -->
  <node id="Inner_Redress_1" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state4"/>
  </node>

<!-- State "Inner_FinalSteady_1" -->
  <node id="Inner_FinalSteady_1" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state5"/>
  </node>

<!-- State "Main_Conflict_1" -->
  <node id="Main_Conflict_1" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state6"/>
  </node>

<!-- State "Inner_InitialSteady_2" -->
  <node id="Inner_InitialSteady_2" type="STATE" data_type="INTEGER">
    <script id="narr.js" func="state7"/>
  </node>
</node>
<node id="Main_Conflict_3" type="STATE" data_type="INTEGER">
  <script id="narr.js" func="state20"/>
</node>

<!-- State "Main_Redress_3" -->
<node id="Main_Redress_3" type="STATE" data_type="INTEGER">
  <script id="narr.js" func="state21"/>
</node>

<!-- State "Main_FinalSteady_1" -->
<node id="Main_FinalSteady_1" type="STATE" data_type="INTEGER">
  <script id="narr.js" func="state22"/>
</node>

<!-- Connect the States -->

<!-- "Main_InitialSteady_1" to "Inner_InitialSteady_1" -->
<edge id="Edge_1" type="TRANSITION" begin="Main_InitialSteady_1" end="Inner_InitialSteady_1" data_type="BOOLEAN">
  <script id="narr.js" func="state1toState2"/>
</edge>

<!-- "Inner_InitialSteady_1" to "Inner_Conflict_1" -->
<edge id="Edge_2" type="TRANSITION" begin="Inner_InitialSteady_1" end="Inner_Conflict_1" data_type="BOOLEAN">
  <script id="narr.js" func="state2toState3"/>
</edge>

<!-- "Inner_Conflict_1" to "Inner_Redress_1" -->
<edge id="Edge_3" type="TRANSITION" begin="Inner_Conflict_1" end="Inner_Redress_1" data_type="BOOLEAN">
  <script id="narr.js" func="state3toState4"/>
</edge>

<!-- "Inner_Redress_1" to "Inner_FinalSteady_1" -->
<edge id="Edge_4" type="TRANSITION" begin="Inner_Redress_1" end="Inner_FinalSteady_1" data_type="BOOLEAN">
  <script id="narr.js" func="state4toState5"/>
</edge>

<!-- "Inner_FinalSteady_1" to "Main_Conflict_1" -->
<edge id="Edge_5" type="TRANSITION" begin="Inner_FinalSteady_1" end="Main_Conflict_1" data_type="BOOLEAN">
  <script id="narr.js" func="state5toState6"/>
</edge>

<!-- "Main_Conflict_1" to "Inner_InitialSteady_2" -->
<edge id="Edge_6" type="TRANSITION" begin="Main_Conflict_1" end="Inner_InitialSteady_2" data_type="BOOLEAN">
  <script id="narr.js" func="state6toState7"/>
<edge id="TR3X" type="TRACE" begin="F3" end="EX" data_type="STRING"/>
</topology>
</model>

<!-- Simulation Frame Setup -->
<simulation start_time="0" end_time="1000" delta_time="1" cycle_time="4"/>
</MXL>

C.5 Complex Example Javascript

/*F1: gen() function generates input*/

function gen() {
    this.F2 = 1;
}

/*F3: disp() displays the story at the current state*/

function disp() {
    print("display");
    if (this.F2 =="Main_InitialSteady_1") {
        this.EX_OUT = "Frodo Baggins lives happily in the Shire with his uncle Bilbo Baggins.";
    }
    else if (this.F2 =="Inner_InitialSteady_1") {
        this.EX_OUT = "Gandalf arrives in the Shire."
    }
    else if (this.F2 =="Inner_Conflict_1") {
        this.EX_OUT = "Gandalf sees Bilbo use the ring and wonders if it is the ring of power."
    }
    else if (this.F2 =="Inner_Redress_1") {
        this.EX_OUT = "Gandalf reads about the history of the ring, finds that fire will reveal if it is the ring of power, and throws the ring into the fire."
    }
    else if (this.F2 =="Inner_FinalSteady_1") {
        this.EX_OUT = "Gandalf finds that the ring is, indeed, the ring of power and sends Frodo on his journey away from the Shire."
    }
    else if (this.F2 =="Main_Conflict_1") {
        this.EX_OUT = "Frodo must get away from the Shire."
    }
    else if (this.F2 =="Inner_InitialSteady_2") {
        this.EX_OUT = "Frodo begins his journey to Brie. Sam, Meri, and Pippin join him."
    }
    else if (this.F2 =="Inner_Conflict_2") {

this.EX_OUT = "The hobbits encounter a ring wraith."
};
} else if (this.F2 =="Inner_Redress_2") {
this.EX_OUT = "The hobbits hide and run, evading the ring wraith."
};
} else if (this.F2 =="Inner_FinalSteady_2") {
this.EX_OUT = "The hobbits make it safely to the Inn of the Prancing Pony in Brie."
};
} else if (this.F2 =="Main_Redress_1") {
this.EX_OUT = "The hobbits make it safely away from the Shire, but arrive in Brie to find that Gandalf is not there."
};
} else if (this.F2 =="Main_Conflict_2") {
this.EX_OUT = "Frodo encounters a dangerous-looking man known as Strider."
};
} else if (this.F2 =="Main_Redress_2") {
this.EX_OUT = "Strider turns out to be a friend and leads the hobbits away from Brie to Rivendell."
};
} else if (this.F2 =="Inner_InitialSteady_3") {
this.EX_OUT = "Strider and the hobbits journey to Rivendell."
};
} else if (this.F2 =="Inner_Conflict_3") {
this.EX_OUT = "The group is attacked by ring wraiths while cooking dinner on the journey."
};
} else if (this.F2 =="Inner_Redress_3") {
this.EX_OUT = "Strider fights the wraiths off, but Frodo is stabbed by one of the ring wraiths, poisoning him. Strider and the group pick up an injured Frodo and continue the journey to Rivendell."
};
} else if (this.F2 =="Inner_Conflict_4") {
this.EX_OUT = "The condition of Frodo gets much more serious and the group must stop."
};
} else if (this.F2 =="Inner_Redress_4") {
this.EX_OUT = "Arwen, an elf, comes to take Frodo to Rivendell faster than the group can."
};
} else if (this.F2 =="Inner_FinalSteady_3") {
this.EX_OUT = "Arwen takes Frodo to Rivendell. The group makes it to Rivendell shortly after."
};
} else if (this.F2 =="Main_Conflict_3") {
this.EX_OUT = "Frodo finds out that the ring must be taken to Mordor."
else if (this.F2 == "Main_Redress_3") {
    this.EX_OUT = "A fellowship is created, and they journey to Mordor. Along the way, the fellowship is broken, but Frodo continues."
} 
else if (this.F2 == "Main_FinalSteady_1") {
    this.EX_OUT = "Frodo gets to a place where Mordor is visible in the distance."
}
else {
    print("You should not ever get here.");
}

/*F2: ------------------ STATES -------------------*/

function state1() {
    out_F2_Main_InitialSteady_1 = 0;
}

function state2() {
    out_F2_Inner_InitialSteady_1 = 1;
}

function state3() {
    out_F2_Inner_Conflict_1 = 2;
}

function state4() {
    out_F2_Inner_Redress_1 = 3;
}

function state5() {
    out_F2_Inner_FinalSteady_1 = 4;
}

function state6() {
    out_F2_Main_Conflict_1 = 5;
}

function state7() {
    out_F2_Inner_InitialSteady_2 = 6;
}
function state8() {
    out_F2_Inner_Conflict_2 = 7;
}

function state9() {
    out_F2_Inner_Redress_2 = 8;
}

function state10() {
    out_F2_Inner_FinalSteady_2 = 9;
}

function state11() {
    out_F2_Main_Redress_1 = 10;
}

function state12() {
    out_F2_Main_Conflict_2 = 11;
}

function state13() {
    out_F2_Main_Redress_2 = 12;
}

function state14() {
    out_F2_Inner_InitialSteady_3 = 13;
}

function state15() {
    out_F2_Inner_Conflict_3 = 14;
}

function state16() {
    out_F2_Inner_Redress_3 = 15;
}

function state17() {
    out_F2_Inner_Conflict_4 = 16;
}

function state18() {
    out_F2_Inner_Redress_4 = 17;
}

function state19() {

out_F2_Inner_FinalSteady_3 = 18;
}

function state20() {
    out_F2_Main_Conflict_3 = 19;
}

function state21() {
    out_F2_Main_Redress_3 = 20;
}

function state22() {
    out_F2_Main_FinalSteady_1 = 21;
}

/*----------------------------------------------------------- EDGES ---------------------------*/

function state1toState2() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 1 to state 2");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state2toState3() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 2 to state 3");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state3toState4() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 3 to state 4");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state4toState5() {
    if(parseInt(this.INPUT) == 1) {
print("transition from state 4 to state 5");
this.OUT = true;
}
else
this.OUT = false;
}

function state5toState6() {
if(parseInt(this.INPUT) == 1) {
    print("transition from state 5 to state 6");
    this.OUT = true;
}
else
    this.OUT = false;
}

function state6toState7() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 6 to state 7");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state7toState8() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 7 to state 8");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state8toState9() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 8 to state 9");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state9toState10() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 9 to state 10");
    }
this.OUT = true;
}
else
this.OUT = false;
}

function state10toState11() {
if(parseInt(this.INPUT) == 1) {
print("transition from state 10 to state 11");
this.OUT = true;
}
else
this.OUT = false;
}

function state11toState12() {
if(parseInt(this.INPUT) == 1) {
print("transition from state 11 to state 12");
this.OUT = true;
}
else
this.OUT = false;
}

function state12toState13() {
if(parseInt(this.INPUT) == 1) {
print("transition from state 12 to state 13");
this.OUT = true;
}
else
this.OUT = false;
}

function state13toState14() {
if(parseInt(this.INPUT) == 1) {
print("transition from state 13 to state 14");
this.OUT = true;
}
else
this.OUT = false;
}

function state14toState15() {
if(parseInt(this.INPUT) == 1) {
print("transition from state 14 to state 15");
this.OUT = true;
}
else
    this.OUT = false;
}

function state15toState16() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 15 to state 16");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state16toState17() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 16 to state 17");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state17toState18() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 17 to state 18");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state18toState19() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 18 to state 19");
        this.OUT = true;
    }
    else
        this.OUT = false;
}

function state19toState20() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 19 to state 20");
        this.OUT = true;
    }
}
else
    this.OUT = false;
}

function state20toState21() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 20 to state 21");
        this.OUT = true;
    } else
        this.OUT = false;
}

function state21toState22() {
    if(parseInt(this.INPUT) == 1) {
        print("transition from state 21 to state 22");
        this.OUT = true;
    } else
        this.OUT = false;
}
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

John Philip Hays was born in Houston, Texas, on October 7th, 1977. Over the next few years, his family moved around Texas, living in such cities as Wichita Falls, Marshall, and Vernon. In 1987, John and his family moved to Oklahoma City, where John spent the next eight years of his life. In 1995, John and his family moved to Port Orange, Florida, a suburb of Daytona. John completed his senior year of high school in Port Orange, and went on to Embry-Riddle Aeronautical University as an aviation computer science major.

A year later, John moved with his family to Atlanta, Georgia, where he finished his undergraduate degree in computer science at Georgia State University. In the spring of 2001, John became the first graduate student accepted into the new Digital Arts and Sciences Program in the College of Engineering at the University of Florida. In May of 2003 John will receive his Master of Science in digital arts and sciences.