`commanimation`: Creating and managing animations via speech

Hana Kim, Nancy Kho, Emily Yan, and Larry Rudolph
Computer Science and Artificial Intelligence Laboratory
MIT
Cambridge, MA 02139
email: rudolph@csail.mit.edu

***Abstract*— A speech controlled animation system is both a useful application program as well as a laboratory in which to investigate context aware applications as well as controlling errors. The user need not have prior knowledge or experience in animation and is yet able to create interesting and meaningful animation naturally and fluently. The system can be used in a number of applications ranging from PowerPoint presentations to simulations to children's storytelling tools.**

I. Introduction

This paper describes a system, called `commanimation`, that allows a user to create and control interesting and meaningful animations in real-time by using speech. It is an outgrowth of the basic work by H. Kim [17]. Animation is a very effective way of delivering information. One can use animations in a variety of applications ranging from simulations to PowerPoint presentations to children's storytelling tools. Creating or controlling animations, however, is still considered to be a specialized task reserved for professionals. We are not interested in producing the state of the art animations that require an exorbitant amount of computing power. Rather, we are interested in the process of creating animations and inventing new tools to help novices create and control animation effortlessly. The focus of this work is on inventing a simple, yet versatile method of animation that allows for interactive control. In particular, we present a speech-driven system that allows one to create and control animations in real time. The system also provides traditional computer inputs – keyboard and mouse, as fall-back methods because the speech interface is still not quite robust to stand alone.

The user can be involved in as much or as little as he wants in the process of creating and controlling animation. In the default setting, the user is given predefined objects and speech commands limited to creating the background and animating characters. However, it is possible to create new characters, as well as extend and modify the speech commands. Furthermore, the speech interface allows the user to create animation in real time – a truly novel approach to animation control.

Any type of "serious" animation control system requires the user to do a significant amount of work (usually scripting or key-framing), if not all of it, beforehand. They draw a clear distinction between the creating period and the showing period whereas our approach is to fuse the two together. Creation of a character is like a macro in that it can be built up from components.

The key feature of the `commanimation` system is that it extends the traditional point and click control by enabling the user to apply an operator to an object along with a set of operands. This is significantly different from the usual model of pushing play, fast forward, reverse, and stop. Hypertext is a step beyond in which a click on a particular spot causes some animation to happen. Of course, much more interactive models are supported, but they tend to be geared towards building an animation of the previous sort. However, it is awkward to specify many operands with only a mouse. That is why we suggest to use the more natural speech interface.

The motivation of our approach came more from the speech side. Currently, speaker independent, domain specific speech recognition is fairly good but it is not perfect. The errors tend to be very frustrating. However, one class of people are not bothered by the mistakes. In fact, they relish them. Little children find it very funny when the computer makes a mistake. Of course it must be the right type of mistake.

The `commanimation` system is described in three sections. The animation infrastructure and basic operations are explained in Section III and the speech recognition component is explained in Section IV. Section V describes how to combine the two into a full system.

II. Related Work

Our system is a part of the research in Intelligent Environments (IE) whose goal is to involve computers in tasks in the physical world and to "allow people to interact with computational systems the way they would with other people: via gesture, voice, movement, and context [6].” IEs are designed to make human-computer interaction (HCI) seamless and computers invisible. Instead of relying on traditional computer UI primitives like menus, mice, and windows, IEs utilize gesture, speech, affect, and context [6][7].

There are interactive animation systems, e.g. Alice, a 3D graphics programming environment developed at University of Virginia and Carnegie Mellon University, and Stagecast Creator, a simulation toolkit developed by Stagecast Software that are similar in spirit to our work.

Alice is a 3D graphics programming environment designed for people who do not have any 3D graphics or programming experience [8]. In particular, they choose a target audience, non-science/engineering undergraduates, and design the system with their needs in mind. Authoring in Alice takes two steps: creating an opening scene and scripting [8]. Similar to our system, Alice’s users choose objects from an object gallery. Unlike our system, Alice’s library contains a large number of low-polygon models. The user can also import objects in popular file formats [8]. The user creates the opening scene by placing the objects in desired locations, setting the camera location, and finally saving this initial state into a world file[8]. Once the opening scene is ready, the user can begin scripting in Python. Alice is fundamentally different from our system in two ways. The obvious difference is that Alice deals with 3D animation while we are mainly interested in 2D animation. More importantly, Alice is essentially a script-based animation control system.

Stagecast Creator is a simulation toolkit that allows children and other novice programmers to build interactive stories, games, and simulations without syntactic programming languages. The goal of Creator is to enable children and novice programmers to construct and modify simulations without a programming language [12]. They accomplish this goal by using two technologies: programming by demonstration (PBD) and visual before-after rules. Users create characters and specify how the characters are to behave and interact [11]. First, users either choose characters from the given list or design their own characters. Then, they create simple rules that govern how the characters move by demonstrating the before and after states. Rules can also manipulate the characters’ properties and appearances. Users can make multiple instances of the same type character. Every instance will have the same set of properties, appearances, and rules; however, it has its own value for each property and its own drawing for each appearance. Extensive user studies have shown that Creator’s scheme, as opposed to that of traditional programming languages or scripting languages, is effective and well-received by novice programmers [11][12].

III. The Animation Infrastructure

The main goal is to create a speech-driven interface for interactive animation control. There is a spectrum of user control. At one end, the user merely plays the role of an audience member using only the play and stop buttons. At the other end, the user is responsible for creating and controlling the majority, if not all, of the action. We group animation control systems into three different categories: the passive user system, where the user is an audience member and nothing more, the limited control system, and the interactive system. Most web animations are examples of the limited control system, providing the user to control a small portion of the animation through mouse clicks or buttons. The scope of control the user has in such systems is carefully limited by the author.

`commanimation` is distinguished from most interactive systems by its speech interface and real-time control. Most interactive systems ask the user to write scripts, specify key frames, or define behaviors in advance [4]. Furthermore, the user is expected to have some level of animation knowledge and experience beforehand. `commanimation` instead, invites the the user to learn simple speech commands created with novices in mind. More importantly, with the speech interface, the system can be quick and responsive. It is clear how the speech-based system can be faster in responding to the user than the script-based system for example. Its quick response time allows the system to be real-time. In the real-time control model, animation takes place as the user speaks into the system.

We build upon a commercial animation system, Flash. Flash is a vector-based animation tool that is widely used for gaming consoles, advertisements, user interfaces, web games, and cartoons. Because it is vector-based, Flash produces animations that are scalable without compromising clarity and resolution [16]. In addition, Flash is quite suitable for interactive applications, and therefore quite applicable to our task at hand. What makes Flash a powerful development tool is its accompanying scripting language, ActionScript. ActionScript is an object-oriented scripting language based on JavaScript. We use ActionScript to manipulate characters and background objects and to communicate with a server which, in turn, communicates with the speech interface.

While we need not concern ourselves with most of the details of the Flash authoring environment, it is important to understand a very important feature of Flash – movie clips. Movie clips are self-contained movies that run independently of the main timeline [16]. Movie clips are ideal for controlling a number of independent objects. They come with a number of properties and built-in methods one can manipulate easily. Movie clips are the fundamental building blocks of our animation. Although end users do not have to deal with them directly, our animation is essentially a collection of movie clips.

There is nothing unusual about the animation system. All the action occurs on the stage or screen. For simplicity, we refer to everything that can be displayed on the stage as an object. The object can be a character or a thing and it can be animated or static. Imagine a dog running down a tree-lined street. The street is a static object. The trees swaying in the wind is also an object but one that is being animated. It is a movie clip that is run over and over again. The dog is also an object. The animation shows the dog's legs moving and the whole object is moved across the stage.

Movie clips are produced by some means that are irrelevant to this system, although `communimation` itself can be used to create movie clips from a set of very primitive clips. The system need only be able to play them and move them along the stage. We refer to these as *simple objects*. There also exist *composite objects*, which are composed of multiple objects both simple and composite, along with a way of animating them. They will be discussed after simple objects are discussed in more detail.

A. Simple Objects

Each simple object is just a movie clip. Each simple object is specified as a file. Objects have several attributes. An object has an initial state, which is just an image that is the first frame of the clip. It also has a final state, which is the last frame of the clip. It can be animated, which is accomplished by cycling through the frames in the clip. Usually, the animation continually cycles through the frames. An object has a boundary which is usually the outline of the object. This is useful so that it does not obscure objects behind it as it moves across the stage. There are other temporal attributes for an object. Some exist only during the actual running of the system whereas others reside in a database and can exist across multiple runs of the system.

B. Operations on Objects

There are only a few basic built-in operations on an object. An object can be `created`. That is, there is an instance of the object created during runtime. There may be many instances of an object. Each instantiation has its own set of attributes.

An object is instantiated at a particular location on stage. The first frame of the object is visible on the stage at that location and it covers any other objects that might be in the same location. It is created in front. For simplicity, it is assumed that the stage is of fixed dimension and precision. It is possible for multiple instances of an object to all have the same location. However, since objects are opaque, it is not possible to tell if there is one or more instances of an object at the same location. The file name of the object is used to specify the object to be created. Since each active object has a current location, the object acts as a synonym for that location.

When an object is created, it is active and becomes the object of naming. It can be given a *stage name*. It is often easiest to name an object just after it has been created, although it can happen at any time an object is active. An instantiated object need not have a stage name, as its file name can be used. The stage name must refer to only one instantiated object. If the name refers to one instantiation and a new instantiated object is assigned the same stage name, then the old association is removed. That is, the stage name refers to the latest naming.

An instantiated object can be `destroyed`. Destroying an object removes it from the stage. The file is not removed. If an object has a stage name, that name can be used to specify the instance to be destroyed. If the file name is used to specify the object, then the last instantiation is destroyed. When an object is destroyed, its stage name is remembered and continues to be associated with the object, or more precisely, with the filename of the object. But this puts no restrictions on the stage name and the name can be reused to refer to an instance of a different object.

The system maintains a database mapping stage names to objects. When an object is created, the system attempts to assign a default name to this new instance. This is only done if the stage name is not already in use. A new stage name can be assigned. The default name is just there as a convenience. If an object is instantiated, given a stage name, destroyed, and then later created again, it will take on the same stage name it was originally assigned. As a side comment, note that there is no need to provide a hide operation since it is the same as destroying and recreating an object. A recreated object will inherit the same stage name and location, provided the name is still available.

This default naming leads to some interesting issues when there are multiple stage names associated with the same object. It is not clear which to associate with the object and using the latest stage name does not work in all situations. The best solution appears to let the context disambiguate. That is, the system makes note of the set of objects on the stage at the time of the creation of an object. The mapping of the stage name to the object is further augmented with these other objects. When an object is created, the system chooses the stage name that was given to the object when the stage was most similar. Extensive experimentation is required to evaluate this decision.

Two other basic operations are to play or `animate`

Operation	Operands	Comment
Create	file	This is either a subdirectory or non-action movie clip
Destroy	object	The file is left unchanged, and attributes are remembered.
Start Animating	object	Sometimes refereed to as performing
Stop Animating	object	This happens at the end of a move
Start Moving	obj & loc	Two parameters are required
Stop Moving	object	
Name_It	file	most recent instantiated
Grow	object	optional parameter of the amount
Shrink	object	optional parameter of the amount
Animate Faster	object	optional parameter of the amount
Animate Slower	object	optional parameter of the amount
Move Faster	object	optional parameter of the amount
Move Slower	object	optional parameter of the amount
Start Building		Start recording actions for new compound object
Freeze_N_Name	stage_name	the object is all action since last start build cmd
Stop Building	stage_name	same as freeze-name operation
Start Reverse		Proposed new operation that undoes the
Stop Reverse		operations so far executed

Fig. 1. The table lists the 18 basic operations. The top group are operations on objects, the second group are attributes of objects, the third group is for building compound objects, and the bottom group are new proposed operations.

an object and to `stop` animating an object. The system continually cycles through the frames of the object until it is given the operation to stop. Associated with the animate operation are `animate faster` and `animate slower`.

The final operation is to `move` an object from its current position to another position. While it is moving, the clip animation is played. It stops playing when it reaches the destination. Associated with move are the operations `move faster` and `move slower`. Once again, the speed for the object is remembered and learned in just the same way a name is learned. There is a special infinitely fast speed, which just causes the last frame to be displayed. Similarly, there is an infinitely slow speed which never lets the object's animation get past the first frame. Locations can be specified by the absolute Cartesian coordinate system and at the very beginning this is necessary. However, once there have been objects put on the stage, it is then possible to use these active objects to specify locations. Also note that it is possible to maintain a model of the stage outside of Flash and then convert higher level commands to the predefined operations. For example, rather than saying move an object to location (x,y), it is possible to map relative locations to absolute ones by remembering the absolute locations of all objects. So, a "move left three steps" command can be converted to "move to (10,13)" if it is known that the object is at location (10,16).

C. Composite Object and Freezing

A sequence of operations, can be frozen at any time. It then becomes a composite object and is subject to all the same operations as an external object. In fact it is remembered beyond the session. The name it is given becomes its external name, which acts just like a file name. In fact, a file is created that appears like a movie clip, but is just a sequence of `communimation` operations. These operations include create, move, animate, etc. All the operations from the `Start Building` command and up to the `Freeze_N_Name` command are joined together to form the composite object.

We are exploring the feasibility to add a `Reverse` operation that allows one to undo the effects of operations. Then it becomes easier to build composite objects from other composite objects. That is, one builds a composite object and gives it a name. Afterwards a new copy of this object is created and reversed part way. Then additional commands can be added and the object frozen to create a new composite object.

D. File Name Conventions

The clips are named with a specific format and placed in a specific directory. A speech domain and an animation domain are automatically generated from the clips in the directory.

The idea is that there is a set of clips associated with an object. There can be and often there are many actions associated with an object, and each corresponds to a movie clip. Each action can be animated by simply

One directory	Directory	Subdirectories
Penguin	Penguin	
Dog	Dog	
Dog.bow		bow
Dog.dance		dance
Dog.eat_ice_cream		eat_ice_cream
Dog.hop		hop
Dog.lick		lick
Dog.run		run
Dog.sit		sit
Cat	Cat	
Cat.bow		bow
Cat.eat_ice_cream		eat_ice_cream
Cat.run		run
Cat.hop		hop
Cat.purr		purr

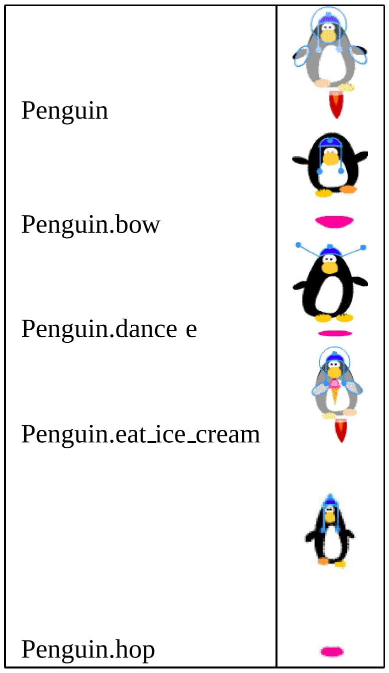

Fig. 2. Examples of how clips are stored in either a single directory or in a directory with subdirectories for each object.

playing the clip or by playing the clip while the object is moving. There are many ways to specify the set of actions to the system. Given that the actions themselves must be manually generated, there needs to be a separate clip or file associated with each action. Any configuration or data base will map the action to the clip. We avoid the need of creating and maintaining such a secondary mapping by using the natural mapping of the file name to the file. That is, the file name itself specifies the object and the action.

The files containing the clips can be in organized in one of two ways. In the first way, each object has a subdirectory and in the subdirectory are clips named according to the actions. In the second way, all the clips are in one directory and the file names consists of two parts: the first is the name of the object and the second is the action with a period separating the two fields. Multi-word actions are accomplished with an underscore, for example `Dog.eat_ice_cream`. Synonyms for actions just become file links and synonyms for objects can be directory links.

IV. Speech Recognition and Processing

Speech recognition systems are classified as being either speaker dependent or independent and as either domain dependent or independent. A speaker-dependent, domain-independent system requires training by the speaker. Once trained, nearly every word spoken is recognized and one has a a dictation system. This leaves the task of interpreting the words and sentences to some other component. A speaker-independent, domain-specific system, on the other-hand, requires the creation of a domain. A domain is the set of words and sentences that are recognized. In other-words, a domain is the grammar accepted by the speech recognition system. The grammar can incorporate much of the semantics of the sentence needed by our animation component. We use this latter approach.

SpeechBuilder is a development tool that allows developers to build their own speech-based applications without delving into human language technology (HLT) [14]. SpeechBuilder asks developers to define necessary semantic concepts by defining *actions* and *keys* to create a domain. Developers can do this either by using SpeechBuilder's web interface or by uploading an XML file and generating domain files based on it. We choose to do the latter to allow frequent changes to the XML file. SpeechBuilder uses an example-based method to allow developers to lay out the specifics of the domain [15]. One needs to provide example sentences for actions along with examples for keys. The tables in Figure 3, taken from the SLS group website, show example keys and actions. In our case, actions would consist of create, delete, move, etc. Keys would consist of object, object_name, etc.

SpeechBuilder provides developers with several ways to write applications, but we make use of the "FrameRelay" approach. That is, the output of the speech recognition component can be relayed to some other component, in our case the Flash subsystem. The processed data is a record of the operators (actions) and operands (keys) that are part of the sentence in the grammar that most closely matches what the user said.

The goal of speech recognition is to find the sentence in the grammar that most closely matches the sentence uttered by the speaker. The set of spoken sentences is much larger than the grammar. The actual spoken sentence is not so important. All that is needed is sufficient information to carry out an operation in the animation domain. It is not important if the user says "Can you please name the dog Fido," "The dog's name is Fido," or even "Call it Fido." What is important is to apply the operation "Name-It" to the explicit operand "Fido" and the implicit operand "active dog."

The basic speech recognition steps are as follows. The audio stream is first divided into sentences. A sentence is delineated by a long period of silence. The sentence is then divided into a set of possible phoneme decompositions. This decomposition is ambiguous and there are usually several possible decompositions. Each decomposition is given a ranking based on how closely the audio wave corresponds to the phonemes. Each of the top n choices are then converted to a sequence of words. Some of the phoneme decompositions can be ruled out at this phase. The words are then analyzed to see if they form a somewhat grammatical sentence by a natural language parser. Finally, the top choices are ranked based on how likely they match a sentence in the grammar. Each sentence in the grammar has an associated XML description that selects the operators and operands in the sentence. So, the final output is a set of XML descriptions each with a given score. There are times that a non-top ranked score is more likely to be correct.

The structure of the XML file describing the grammar is straightforward. There are actions and there are operands. For the most part the operands are either the objects or a location. An object is either a file name, a stage name, or absolute stage location. Instantiated objects are also location specifiers since each instantiated object has an associated location. Objects can also be the subject of a sentence, as in "Fido, fetch the bone." Both "Fido" and "bone" are objects. Actions either move or play an animation. "Fetch" is an animation and a move. It requires the playing of the fetch movie clip. The destination of the move is at the location of the bone.

The XML description of the command is passed to the animation system to carry out the operation. Some operations may not be relevant in the current state of the animations. For example, there may be no dog on the stage. The top several choices are examined in sequence to find one that matches. If none match, then some negative acknowledgment is sent back to the user.

The speech domain is automatically generated by the system. Except for the stage names, all information is gathered from the clip files. The object file names form the set of valid objects and the action file names form the set of valid operators. Each domain also contains a set of basic sentences for creating, destroying, and applying other basic operations. The database of previous stage names is also used to define a set of objects. Finally, a set of well known stage names are also included in the domain.

What is missing from the domain are new, never seen before stage names. We propose to augment the system with the potential for the user to train the system. Unfortunately, with the way recognition works, it is not possible for the system to say that there is a word in the sentence that is a new stage name. The system will either ignore the new word or try to map it to an existing word in the domain. When such mistakes happen, it would be useful to have the user augment the domain.

V. Putting it all together

Given a directory with movie clips in the correct format and naming convention, the application is automatically compiled. The objects and actions are collected with the files in the directory. This information is used to build the speech domain and the animation domain. The speech domain has a set of defaults and alternate ways of saying the basic actions such as create and destroy. For example, "Make a penguin" or "I want a new penguin" will both create a penguin.

A. *Generating Speech Domain Files*

The smart object naming scheme, keys and actions are used to generate speech domain files. The GALAXY framework, the architecture for dialogue systems used by SpeechBuilder, utilizes speech domain files to understand user input and produce desired output. Domain files are generated based on an XML file that contains all the necessary information to create a speech domain – keys and actions. The XML file is easily generated using the object naming scheme.

A text file is created that specifies the names of the movie clips created for all objects. Only one file is needed for all objects. A sample text file would read:

```
Begin Penguin
  Penguin.bow
  Penguin.dance
  Penguin.eat\_ice\_cream
  Penguin.hop
End Penguin
```

If one wants to add more objects, he can add them right after the Penguin movie clips. There are three

Action	Examples
Identify	What is the forecast for Boston What will the temperature be on Tuesday I would like to know today s weather in Denver
Set	Turn the radio on in the kitchen please Can you please turn off the dining room lights Turn on the TV in the living room
Good_bye	Good bye Thank you very much good bye See you later

Actions	Keys
Identify	Atlanta, Boston, Baltimore, ...
Set	temperature, weather, ...
Goodbye	Monday, Tuesday, ...

Fig. 3. Examples of "actions" in SpeechBuilder knowledge representation (left table) and of actions and keys (right table).

simple rules in writing the text file. We need to place one movie clip per line. The default movie clips does not need to be listed. All the movie clips for an object need to be grouped together, begin with "Begin Object" end with "End Object." For example, "Dog.bark" can not appear in the middle of the Penguin movie clips. It needs to appear somewhere between "Begin Dog" and "End Dog."

Because of the intuitive naming scheme, the XML file can be easily generated based on the text file. The XML file is grouped into a number of blocks. The header of each block is generated based on actions and keys. The text file is used to fill in the values of keys and generate example sentences for actions. For example, task is a key and consists of tasks objects can perform. The XML file would contain the following lines for task:

```
<class type="Key" name="task">
  <entry>bow</entry>
  <entry>dance</entry>
  <entry>eat\_ice\_cream</entry>
  <entry>hop</entry>
</class>
```

For an action, entries would consist of example sentences. For the action `create`, the following lines would be generated:

```
<class type="Action" name="create">
  <entry>create a penguin</entry>
  <entry>make a penguin</entry>
</class>
```

It is important to note that the above segment not only specifies two different ways of invoking `create` but also indicates that it can be applied to all objects. In other words, using one member of the key in an example sentence is sufficient to indicate that the action applies to all members of the key. Here's another example:

```
<class type="Action" name="perform">
  <entry>make Bob dance</entry>
</class>
```

Here, "dance" is a member of the key "task." Therefore, the entry above indicates that users can say, "make Bob bow," "make Bob eat ice cream," and "make Bob hop." As a result, the XML file generated is a concise yet easy-to-understand representation of our speech domain. In addition, if one wants add or modify speech commands for an action, he is welcomed to modify the relevant block of the XML file. For instance, the following block is generated for the action `delete`:

```
<class type="Action" name="delete">
  <entry>delete Bob</entry>
</class>
```

One can easily add another entry to the block:

```
<class type="Action" name="delete">
   <entry>delete Bob</entry>
   <entry>pop Bob</entry>
</class>
```

Next the system must generate domain files. To review, domain files are a collection of files that the Galaxy framework uses to understand and process speech input from the user.

B. Voice input relayed to Flash

FrameRelay, an application written in Java, is used to relay data from SpeechBuilder to Flash. Once the sound input has been parsed by SpeechBuilder, FrameRelay processes this output, extracts the relevant information (i.e. action and subject), builds the appropriate XML node, and sends this to Flash. Using ActionScript, Flash then takes this XML node and executes the command requested. The corresponding visual effect is displayed on screen.

C. Creating the background

In the early stages of development, we provided a ready-to-use background for end-users. This was obviously not a good choice for most users would like to create their own background. We still provide the default background; however, we also give the user an environment to create a background. The background not only sets the look and feel of animation but more importantly contains all the background locations. Although there is no distinction between background objects and characters, they are used in different ways. Objects that are part of the background can be individually referenced but cannot be individually moved. Moreover, there are differences between composite objects that are used as characters and those that are used as background. Further experimentation is needed to fully understand the different constraints. For example, the order of objects is very important in the background, so that the tree is in front of the house but behind the car. This ordering is always important but seems to be a major preoccupation when building the background. Consequently, additional primitive operations may need to be added. In addition, the set of primitive building blocks differs for the background and foreground. It might be helpful to have different libraries for each. When the background is finished being created, the user can "freeze" the background and need not give it a name. Freezing signals that the user is finished with creating the background.

D. Animating characters

Animating characters is not too different from creating the background. Characters are born out of the same set of objects from which background objects are created. Any object added after the freeze command is considered a character. One can create, name, and delete characters the same way he would with background objects. However, one needs to use a different approach to move characters around. Moving a character to a grid point as done with background objects is neither interesting nor meaningful. More importantly, this approach is not practical when the user is trying to produce animation in real time. We have decided that one good approach would be to move a character from one background object to another. This approach reduces the number of commands the user has to speak, thereby resulting in more continuous animation. Furthermore it gives the user full control over characters' actions since he is responsible for placing the background objects.

In addition to moving, characters are capable of performing tasks. When we made objects, we assigned a number of tasks to each object. Tasks are designed to give the user freedom to create more interesting and content-rich animation. One can think of tasks as clip arts for animation only less annoying and more dynamic. For example, Penguin can bow, dance, eat ice cream, and hop. Novices would have to spend a great deal of time and effort if they were to create these effects themselves. We diligently provide objects for novices to create fun and useful animation so that they do not have to delve into the details of cartoon making. Ambitious novices and experts need not worry. They can always choose not to use already provided objects and instead create their own objects as long as they adhere to the naming scheme.

VI. Conclusion

We've created a multi-modal animation control system that lets users create and control animation in real time. Users can design the background and animate characters using the speech interface developed with SpeechBuilder. Users need not learn the specialized animation terms and techniques that other systems require them to master. It is important to note, however, that the user has an option to create their own objects and speech commands if he so desires. This is to accommodate both novices and experts. More importantly, animation happens before their eyes as they speak to the system. Although we have not conducted formal user studies, informal user sessions have shown that users learn to use our system in a matter of minutes, use the speech commands with ease, and most importantly, enjoy using our system to a great extent.

The speech enabled animation system poses many challenges while providing a human-centric approach to controlling an animation system. The system itself is a system of systems, with the speech recognition system running on a dedicated Linux computer, the Flash animation system on a Windows computer, and the audio capture running on a handheld iPaq computer.

Speech recognition is somewhere between 80 to 90 percent accurate. In other-words, there are lots of mistakes. We have begun to address ways in which we can limit the errors. First, all the recognized operations contain scores, and so the system can choose to require high confidence scores for serious operations, such as "terminate," and lower confidence scores for operations that can easily be undone. Second, it is possible, in fact sometimes desirable to allow some mistakes to go through without requiring the user to re-speak the command. The idea of turning a weakness into a feature is intriguing. Consider Figure 2. Suppose the system thought one asked for the cat to dance. There is no movie clip associated with a cat dancing. Rather than an outright rejection, `communimation` does the following search. If there is another active object on the stage that can dance, then that object will dance. If there are no such objects but there is an active cat, then the cat will perform a random actions. Similarly, if the cat is commanded to run to the bone and there is no bone on the stage, then some other object, with a lower recognition score, is chosen as a destination. Finally, rather than telling the user that the system does not understand the command, a random active object is

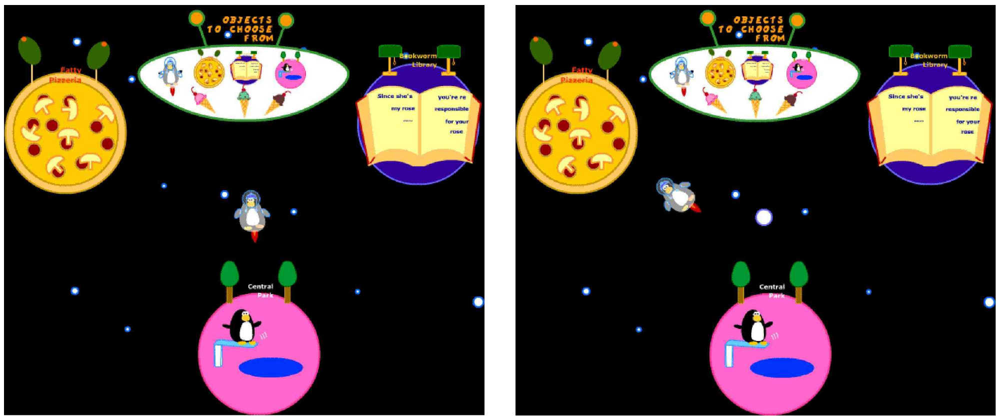

Fig. 4. The user say "Create a penguin." and then "Name it Bob." A penguin is created in front of the existing objects. The next command is "Bob, go to the pizza." which causes the penguin to move from its current position to the pizza. The "go to" movie clip shows rocket fire propelling Bob.

chosen and one of its actions is performed at random. This is where a standard reverse or undo primitive would come in handy.

Our system turns out to be very engaging and plain old fun. We must admit that in our prototype we have deliberately designed the objects and the tasks to be interesting and somewhat humorous to capture the user's attention. When the penguin character became noticeably chubbier after eating an ice cream, most users kept making him eat more ice cream to observe the poor penguin's dramatic increase in size. Most of the test users, mostly graduate students and undergraduate students at Laboratory for Computer Science, exhibited childlike enthusiasm when creating and controlling animation with our system. This is not to say that our system is only designed for light weight, just for fun applications. In fact, a number of practical applications, ranging from PowerPoint presentations to simulations, can be built upon our framework. We interpret the users' enthusiasm and enjoyment as our success in creating something novel and engaging. More importantly, with the test users' encouragement, we are now in a position to make improvements to the system as well as build applications on top of it.

References

[1] Larry Rudolph, *ORG-Oxygen Research Group*. Laboratory for Computer Science Annual Report 2001

[2] Kristin R. Thrisson. *Toonface: A System for Creating and Animating Interactive Cartoon Faces*. Learning and Common Sense Section Technical Report 96-01. April 1996

[3] Daniel Gray, John Kuramoto, and Gary Leib. *The Art of Cartooning with Flash*

[4] ACM SIGGRAPH. http://www.siggraph.org/

[5] Kristinn R. Thrisson. *A Mind Model for Multimodal Communicative Creatures and Humanoides.* Internal Journal of Applied Artificial Intelligence 449-486. 1999

[6] Michael H. Coen. *Design Principles for Intelligent Environments.* Proceedings of the Fifteenth National Conference on Artificial Intelligence. (AAAI'98). 1998

[7] Michael H. Coen. *The Future of Human-Computer Interaction or How I learned to stop sorrying and love My Intelligent Room.* IEEE Intelligent Systems. 1999

[8] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, et al. *Alice: Lessons Learned from Building a 3D System for Novices.* Proceedings of Conference on Human Factors in Computing System. 2000

[9] Matthew Conway. *Alice: Rapid Prototyping System for Virtual Reality*

[10] *Alice v2.0b.* http://www.alice.org

[11] Allen Cypher and David Canfield Smith. *KidSim: End User Programming of Simulations.* Proceedings of Conference on Human Factors in Computing System. 1995

[12] David Canfield Smith, Allen Cypher, and Larry Tesler. *Novice Programming Comes of Age.* Communications of the ACM, 43(3), March 2000, pp. 75-81. 2000

[13] *Stage Cast Software, Inc.* http://www.stagecast.com

[14] Spoken Language Systems, MIT Laboratory for Computer Science. http://www.sls.lcs.mit.edu

[15] Eugene Weinstein. *SpeechBuilder: Facilitating Spoken Dialogue System Development,* Master of Engineering Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 2001

[16] Ethan Watrall and Norbert Herber. *Flash MX Savvy.*

[17] Hana Kim *Multimodal Animation Control*, Master of Engineering Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 2003.