jtap - Using Tcl to design interactive eLearning materials

by Diplom-Medieninformatiker (FH) Christian Kohls, Diplom-Medieninformatikerin (FH) Susanne Kaiser and Diplom-Wirtschaftsinformatiker (FH) Tobias Windbrake / University of Applied Sciences Wedel

Introduction

jtap is an authoring tool for multimedia content, focusing on eLearning materials. With jtap, one can create dynamic and interactive slides using a visual what-you-see-is-what-you-get interface. The program is written in Java using the powerful graphics engine and capabilities of Java Swing. We use Jacl to offer course designers the opportunity to add interactivity to their slides. With Tcl they can define action lists to handle events and perform the communication between learning objects on the board. Also we use Tcl for the synchronization of multiple computers, so that the content of a slide can be displayed simultaneously on many computers at once. Each change to a slide (e.g. moving objects, highlighting or writing) will occur on all connected computers at the same time.

A slide in jtap is like a slide in an average presentation program, e.g. PowerPoint. You can arrange the slides to a curricula along a multi-layered timeline to define the display order. One layer can be used for a master slide, containing objects that will be visible during the complete presentation. Another layer can be used for content that changes after every presentation step. Since the timeline provides unlimited layers, there also could be a layer that changes its content less frequently, e.g. for objects that apply to the scope of a chapter.

Learning Objects Interacted by Tcl

The power of jtap is hidden in the objects that can be placed on the virtual slides. The complexity of an object can vary from simple text and graphic elements up to complete applications (database access, spreadsheets). There are already many standard objects included (textfield, button, graphic, paint area, html, video, sound, etc.), but the set will be further extended by plugins. The appearance and behavior of an object is set by a list of properties. These properties can also be manipulated during the runtime of a presentation or learning course. Of course, these manipulations must first be defined. Here Tcl plays a major role, because all the scripts for a course are defined as Tcl programs. We extended the language with special multimedia commands that can be applied to the jtap objects. There are simple commands to set or get the property value of an object. There are also some commands that define animations or complex work tasks for an object (test evaluation, database connections) within only one line of code.

Script Types

We distinguish between slide scripts and event scripts. Slide scripts belong to a specific slide in jtap. There is one onEnter script that will be executed each time a slide will be displayed for the first (or n-th) time. Here one can define initial actions, e.g. resetting Tcl variables or object properties. Animations to build up the slide content can be defined: objects could slide in one by one from the screen border (an effect well-known from PowerPoint). The onRepeat script will be evaluated continuously again and again. This is useful for complex animation

steps in simulations. Upon leaving a slide, the onExit script will be invoked to store data or analyze the user input.

The second category of Tcl scripts in jtap are event scripts. Those scripts belong to one object instance. Technically the user changes string properties to define a Tcl script for each event that can occur in an object. Objects support the regular GUI mouse and key events. Some objects have properties for uncommon event types: For example there is a multiple choice test object with the events "correct answer" and "false answer".

Applications of Tcl

Applications of Tcl in jtap include animation effects, interaction of objects, navigation structures, tutoring, test evaluation and simulations. For animations one can calculate an animation path for an object based on the property values of other objects. All kinds of objects -- graphics or sub applications -- can be moved on a slide by one simple Tcl command. A wide range of test forms can be evaluated by Tcl scripts. Conditioned expressions can be used to select the next test page or question depending on the previous results of the student. This can also be used for smart navigation through a course. Jumps to other slides of a course can depend on which exercises the student attended before. The visible content of a slide can be specified by the number of times the student used it. Texts and explanations can be more detailed for those who return several times to the same page. Eventually, Tcl scripts can be used to implement simulations or virtual experiments within an eLearning course.

Using Tcl for Animation

There are at least two new powerful commands to create animations: morph and slide. Both commands change a numerical property of a visual object over a specified time period. The syntax is this:

```
morph slideName elementName propertyName propertyValue duration
```

The combination of slideName and elementName specifies the object that is subject to be changed. With propertyName and propertyValue one specifies a new value for a certain property. However the value is not set immediately, but will be approximated over the time period that is defined in duration. If we have a simple graphic object named "worldGraphic" which is on the slide "demonstration" we can move the object from its current location to a new x position:

```
morph "demonstration" "worldGraphic" x 80 12
```

Here we move the object to the horizontal screen position 80. The movement will take 12 time steps. By changing the location properties you can animate any kind of a visual object. Even a text field or a spreadsheet could be moved on the screen using the morph command. Animation can be applied to each numerical property that effects the visual presentation on the screen. Examples are properties for colors, opacity, object size, rotation etc.

By embedding the morph command in a Tcl script, you can define dynamic animation behaviours. For example, you can use a variable that defines the speed of the animation:

```
set animSpeed [expr $userSettings * $someotherData ]
...
morph "demonstration" "worldGraphic" x 80 $animSpeed
```

In the previous example the animation speed (that is the number of time steps needed to approach the property's target value) depends on the variable's userSettings and someotherData. Animations could also depend on conditions:

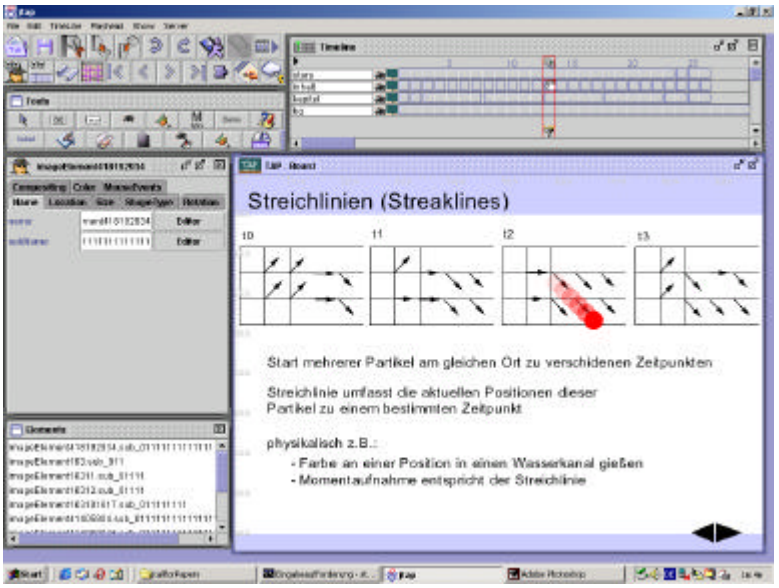
```
set stockValue [expr $stockValue + rand() * 10 - 5]
if { $stockValue > 100 } {
     morph "demonstration" "worldGraphic" x 90 12
     morph "demonstration" "worldGraphic" y  0 12
} else {
     morph "demonstration" "worldGraphic" x 0 12
     morph "demonstration" "worldGraphic" y 90 12
}
```

Here we moved our "worldGraphic" into the upper right corner of the screen, if the stockValue is greater than 100. Otherwise, the "worldGraphic" would move into the lower left corner.

Assuming we have another graphic object "sunGraphic" on the slide and we want the "worldGraphic" to move towards the "sunGraphic". Then we can write this code:

```
morph "demonstration" "worldGraphic" x [getProperty "demonstration" "sunGraphic" x] 12
morph "demonstration" "worldGraphic" y [getProperty "demonstration" "sunGraphic" y] 12
```

The command getProperty returns the current property value of an object. The corresponding setter command is setProperty. setProperty can be used to directly change a property value instead of animate it.



Using the morph command we animate the red particle from its original position to a new destination:

```
morph $thisSlide particle x 67
morph $thisSlide particle y 37
```

Test Evaluation and User Feedback

In a Tcl script we can evaluate the current state of a jtap slide and respond to user activities. If we have a slide with questions and text fields for the answers, we can define an event script for a button object. In this script we check the user input and can give feedback by changing the displayed slide:

```
if { [getProperty "demonstration" "inputField" text] == "User answer" } {
  setProperty "demonstration" "infoLabel" text "Yep, the answer was right."
  setProperty "demonstration" "infoLabel" green 255
  incr correctAnswers
  set score [expr $score + 30]
} else {
  setProperty "demonstration" "infoLabel" text "Oh no! Wrong answer."
  morph "demonstration" "infoLabel" red 255 12
  incr wrongAnswers
}
```

In the previous example we type-matched the input of the text field "infoLabel" with the string "User answer". If both match then we give a positive user feedback. We set the text of "infoLabel" to the string "Yep, the answer is right." In addition we change the text color of the label to red, increment a counter and increase the user's score. If the user's answer does not match, we give a negative feedback and increment the counter "wrongAnswers". The color of "infoLabel" will turn to red. Instead of directly setting the color, we use color animation.

Directly comparing a string to a user input is not very user friendly, because it will not tolerate any kind of errors. The strings "UUser" and "User" should be treated the same way since type errors should be tolerated. Tcl of course allows more flexibility in matching the user input, e.g. using regular expression.

The user may have different ways of manipulating the slide content at the runtime of a learning session. He can enter text into text fields, drag objects on the screen, use check boxes or menus on a slide. In a Tcl script we can interpret the user settings as a test solution. Depending on the test results the Tcl script can store and manipulate variables, change and animate object properties or jump to another slide of the course.

Creating Usertracks

You can track the user path through the slides using a Tcl list. Each time the user leaves a slide the onExit script will be executed. In this script a list of "visitedSlides" can be extended:

lappend visitedSlides $thisSlide

The variable "thisSlide" is set by the jtap environment and contains the name of the script's hosting slide. We can use the list in several condition statements. Assuming the access to the slide "Final exams" requires that the user has seen the slides "Chapter One", "Chapter Two" and "Exercise" before, than we can check this condition in the onEnter script of "Final exams":

```
if { [lsearch visitedSlides "Chapter One"] == -1 || [lsearch visitedSlides
"Chapter Two"] |== -1 || [lsearch visitedSlides "Exercise"]} {
   go slide "Deny Message"
}
```

Using the lsearch command we checked, whether all requested slides are part of the list. If one of the searches failed (lsearch returns -1) then we use the jtap command "go" to jump to the slide "Deny Message".

There is another way to check whether a slide was viewed or not. With the jtap command "slideInfo" you can find out how many times a slide was displayed. You can use that information to hide or show additional information depending on the number of times a slide was used. The multiple use of a slide could be a hint that the user is missing some information. If we define the following onEnter script for a slide, the text object "extraInformation" will show up on the fifth use of the slide:

```
if { [slideInfo $this visits] > 4 } {
  setProperty $this extraInformation visible true
}
```

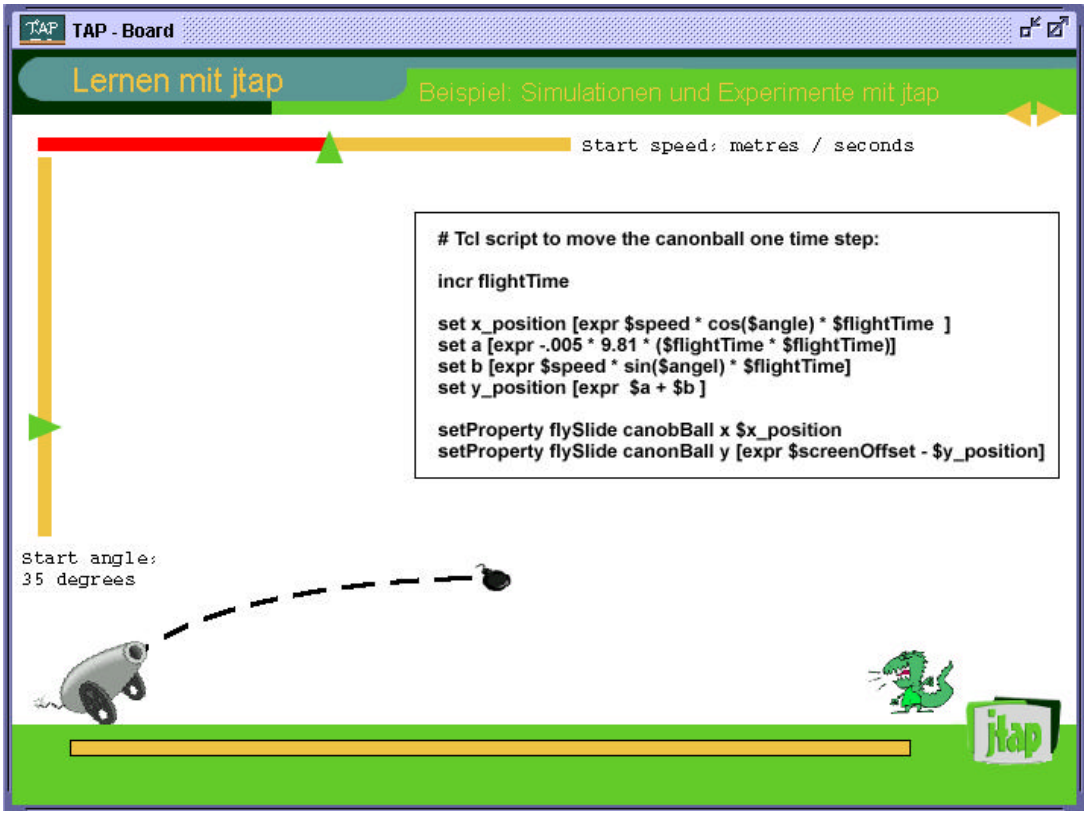
For navigation we can also use more complex conditions considering the results of former tests, the usage of slides or individual user settings. The following script could be an example for the onPressed script of a "Next" button object:

```
if { [slideInfo "Overview of Content" visits] >= 1 && $userLevel ==
"beginner"    && $quizScore > 120 && $result == 30  } {
   go slide "Expert Level"
} else {
   go next
}
```

Implementing Simulations and Experiments

Tcl can add real interactivity to eLearning materials. With a Tcl script a course can react dynamically to user input and direct the behaviour and appearance of objects. One can specify all the control structures for simulations, experiments, business games or case studies with Tcl. The input and output of data will use the infrastructure and graphic interface of jtap. On a slide you can use standard objects for graphics, texts, several test forms (such as multiple choice), video, audio and GUI elements.

The initial setting for an experiment can be defined in the onEnter script of a slide. Here one can set all the required variable values. In the onRepeat script the programmer defines all continuous dynamic changes. As an example a slide could contain an area with a virtual gravity field. Inside the field all objects move according to the physical laws of gravity. Each step of the movement can be calculated in the onRepeat script. Since this script will be executed continuously as long as the slide is shown, the objects handled in the script will move step by step. While the simulation is running the user can manipulate the experiment settings. The event scripts of objects handle the user input and allow the user to drag other objects into the gravity field or change the gravity constant. Finally in the onExit script we can analyse the current position of the objects and store the results in variables to access those within other slides.



Experiments: The onRepeat script defines the cannonball's movement regarding to a physical law. Each time the script is invoked one step of the movement will be executed. The user first can set up the start speed and angle using the sliders. The flight starts when the user presses on the cannon.

Tcl is also used internally

jtap not only offers Tcl as a scripting language for the program user, but also uses Tcl internally for distributed computing and white board functions.

jtap is applicable as a standalone version, but also can run synchronously on several computers.

Every jtap application is provided with an integrated server, that can be started individually. As soon as the server runs, several remote clients can connect immediately.

To preserve consistency in content, the server initially transfers the current presentation to each connected client. After the content has been loaded locally, every networked student shares the same view and working space.

Since we have encapsulated the ways to set the properties of objects, it's easy to monitor each access. At runtime if the elements of the distributed presentation have been modified, the corresponding events will be delivered as messages to each participant. This network protocol consists mostly of simple proprietary Tcl commands. If an interaction occurred, a client-side message will be generated and sent to the server automatically. In this context the server only acts as a mediator by accepting this message, extracting the address data and forwarding the rest of the information to the appropriate receiver. At the receiver the parsing of the

transferred message, consisting of Tcl commands, will be delegated to the Tcl interpreter. It extracts the commands and its parameters. As a result the state of presentation will be updated, corresponding to each connected computer by setting the values of elements' properties respectively. To use the Tcl interpreter, it facilitates the development expenses enormously. We do not have to take care of parsing issues and can extend the network based functionality easily by just adding new Tcl command classes to our systems.

Conclusion

Using Tcl in our project guranteed the use of a well-developed language. The easy-to-use syntax allows every novice to write simple actions, while the more experienced developers are satisfied, too. jtap's event-driven use of Tcl scipts does not limit the use of jtap to eLearning applications but rather provides a way to build graphic user interfaces for Tcl in general. Also, the combined use of Tcl and Java unites the advantages of both worlds.

About the project

jtap was started by a small team of graduated students at the Wedel University of Applied Sciences, Germany. In the meantime, there are about 50 students who work for the core team and extend the software. In the summer of 2002 we intend to publish the sources and provide English documentation alongside the original German version. The complete software can be downloaded at: http://www.jtap.org

Contact

Christian Kohls
Feldstrasse 143
22880 Wedel
Germany
christian.kohls@jtap.org