Javanti
Using Tcl to design interactive eLearning materials

Diplom-Medieninformatiker (FH) Christian Kohls
Diplom-Wirtschaftsinformatiker (FH) Tobias Windbrake
Diplom-Medieninformatikerin (FH) Susanne Kaiser

University of Applied Sciences Wedel
Feldstrasse 143 – 22880 Wedel - Germany

christian@kohls.de

Abstract

Javanti is an Open Source authoring tool for multimedia content, focusing eLearning materials. One can create dynamic and interactive slides using a visual what-you-see-is-what-you-get interface. Slides contain learning objects such as formatted text, animated graphics and several quiz types. Objects can be placed individually on a virtual board. To provide an interactive learning environment those objects should respond to user input and communicate with each other. We needed a scripting language to define the dynamic actions at runtime. Tcl was our first choice, because of its flexibility, extensibility and easy integration into applications. With Jacl there is an implementation of Tcl entirely written in Java which we use in our software product. This paper discusses the purpose of Javanti, explains why we decided on Tcl as an integrated scripting language and shows how Tcl is used to create interactive eLearning materials.

1. About Javanti

1.1. E-Learning Materials Today

The combination of computer-based training with the immediate accessibility of the Internet has made a wide range of new eLearning applications available today to the average corporate trainer and trainee. The successful merging of these two technologies can now be utilized for a variety of different, highly productive training functions including: online lectures, documentary videos, simulations, virtual experiments, case studies and game-based learning. Yet in most cases today, even with this large array of technologically-viable options available, we still only find simple texts and scripts -- often even without the most basic of hyperlinks incorporated -- offered to us under the limited capabilities of a standard browser or presentation program.

1.2. Four Level of Content Creation

1.2.1. Creating slides

With Javanti, we have introduced a 4-level-model of content-creation. In this model, you can create virtual slides and arrange them to different curriculum types. Each of these virtual slides may contain static, dynamic and interactive elements. The model takes respect to the different technical skills of trainers. Basically a slide in Javanti is like a slide in an average presentation program, e.g. PowerPoint. However Javanti's slides are not limited to static content. There is a wide range of interactive objects available by default. In addition course developers may use Tcl to define individual behavior for objects and slides.

1.2.2. Basic Level

Most trainers start on the easiest level by using standard objects in a WYSIWYG editor. These elements can be either static (text, graphic objects, html documents), dynamic (keyframe animation, video, sound) or interactive (smart answer fields, paint areas, multiple choice tests). Without any previous knowledge of the software trainers can design the slides on a virtual white board. Each slide will be organized in a multi-layered timeline. The multi-layer concept allows the combination of more than one slide on the board. That is one can have as many background, foreground, chapter and content layers as he wants. For example you can place a table of content element in a foreground layer that is visible during the complete course whereas the content changes on a different layer each time you navigate to another page. In addition we can use chapter layers containing one slide for each learning episode. The table of content element can be generated automatically. Several alternatives for navigation through the course are supported. For a student's orientation there are many possibilities to indicate the current progress of the course. Students have the opportunity to define their own preferred structuring of the content. On each slide they can write down their own notes and drawings. Beside the core information on a slide you can insert links to additional learning materials. These materials will be available within in a Javanti course for students that want further insights or need more detailed explanations.

1.2.3. Simple actions

The second level allows object interaction with simple Tcl commands. We extended the command set of Jacl with some

multimedia commands. Using these commands one can define property changes, start property animation or jump to other slides. We can define a list of Tcl commands for different events. For example by clicking on a picture we can show or hide other elements, change their size, color or position and start other media resources (video, animation) within the same slide. The usage of Tcl in this level is very simple and does not require any understandings of programming concepts.

1.2.4. Program development

The third level uses the full range of Tcl by adding real functionality to each slide. This is useful for any type of calculation, controlling and macros. The response on test evaluation or mouse events can be defined in small Tcl scripts. Simple programs can be created in an EasyScript mode by drag&drop commands to a command list.

On the fourth and highest level, one can optionally write new object types in Java.

All levels can be combined and imply the suitability for each learning Paradigm. Standard elements can support the learning processes for base knowledge in a traditional way. Simulations, virtual experiments, case studies and game-based learning can be implemented by using Tcl or Java to address higher education.

2. Integration of Tcl

2.1. Requirements for Scripts

2.1.1. Usage of scripts

The power of Javanti is hidden in the objects placed on virtual slides. The complexity of an object can vary from simple text and graphic elements up to complete applications (database access, spreadsheets). 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. There are also some actions that define animations or complex work tasks for an object (e.g. test evaluation, database connections). Obviously these manipulations must be defined somewhere and somehow. To define interactive and dynamic actions, respond to user events and to prepare generated slide content a scripting language is needed. We found some requirements for a scripting language used in course development.

2.1.2. Simplicity

The language must be very easy to learn and use. Most course designers do not have a computer science background. They want to concentrate on educational content rather than bother with program development. In Tcl each statement is a simple line of plain text. Without any knowledge about control and program structures users can define simple action lists. They do not even recognize that they are using a programming language.

2.1.2. Full featured language

Full capabilities of a modern programming language are needed. While some people are frightened by the power of a programming language, the more professional course developer needs a sophisticated language to create simulations, experiments and interactive test forms. Tcl not only satisfies those requirements but adapts itself perfectly to the skills of a user. Non-technical users will never see and use all the commands of Tcl. They are happy with a small subset of powerful commands to delegate work to Javanti objects. Other users can use simple control structures with none or only little knowledge of software design to take more control over Javanti objects. Professional programmers however are pleased to have access to a wide range of standard Tcl commands and can take complete control over Javanti.

2.1.3. Simple syntax

The syntax structure must be simple. In future versions of Javanti our development team plans to offer a visual interface to define action lists. The most common command settings will be available in a graphical user interface. For example Javanti's animation commands appear in a visual box to define all animation parameters by using list boxes and numerical sliders. To provide an adequate representation a very basic structure is needed. Tcl matches this request because of its simple grammar and substitution rules.

2.1.4. Multimedia commands

Multimedia commands must be supported. A lot of course designers like to see a language to create multimedia effects without programming. With Tcl we were able to create new powerful commands and add them easily to the Tcl interpreter. Our multimedia commands operate on all Javanti objects. They can be considered as links between Tcl programs and Javanti objects.

2.1.5. Implementation

The language should be easy to implement. When we started the development of Javanti our human resources were limited. Our goal was to create a presentation tool with powerful animation and assessment capabilities. To develop a new language from scratch in time was simply impossible for us. In addition we never aimed to create just another language to force even the professional users to learn a new syntax and another set of commands. Tcl is a very effective way to add a programming language to any software application. Actually it took us less than a day to integrate Jacl to our program. In just a few hours we extended our product with all the possibilities of a programming language!

2.1.6. Tcl matched perfectly

Tcl was the only programming language on the market satisfying all our needs. When we considered our options there were some issues less important to us. For example we did not care too much about performance. All time-critical actions are performed by Javanti objects on the board. Tcl as a scripting language acts mainly as glue between those objects. Also we did not need a

language which fully supported the object-oriented programming paradigm. For object-oriented development Javanti supports a plugin interface to integrate new objects written in Java. These new objects again benefit from Tcl, since Tcl programs can directly address those objects.

2.2. Usage of Jacl

Javanti is written in Java using its fast graphic engine and GUI capabilities of Java Swing. To integrate Tcl in Java applications Mo DeJong and Scott Redman have enlightened the Tcl community with Jacl, a 100% pure Java implementation of Tcl. There are some minor restrictions in Jacl, e.g. some commands are missing. Fortunately none of them is important to our software.

The integration of Jacl into Java applications in general is very simple and effective. We only had to add two Java Archive Files to Java classpath to integrate the Tcl interpreter. Once an instance of the interpreter class has been created, we can use it to evaluate strings and files containing Tcl scripts.

To extend the command set of the interpreter, we simply created sub classes of a command interface and added single instances to the interpreter:

```
myInterp = new Interp();
myInterp.createCommand("setProperty",
                       new CmdSetProperty() );
myInterp.createCommand("getProperty",
                       new CmdGetProperty() );
myInterp.createCommand("notify",
                       new CmdNotify());
myInterp.createCommand("go" ,
                       new CmdGoPage() );
myInterp.createCommand("playSound" ,
              new CmdPlaySound() );
myInterp.createCommand("morph",
                       new CmdMorph());
myInterp.createCommand("slide",
                       new CmdSlide());
myInterp.createCommand("save",
                       new CmdSave());
myInterp.createCommand("load",
                        new CmdLoad());
…
```

2.3 Script Types

2.3.1. Slide scripts

In Javanti Tcl scripts will be executed if particular events occur. We distinguish between slide scripts and object scripts. Both scripts can be entered in a script editor (see Figure 1). Slide scripts belong to a specific slide in Javanti. For each slide the user can define three different Tcl scripts: onEnter, onRepeat, onExit.

The onEnter script will be executed if a slide becomes visible on the virtual board. Here one can define initial actions, e.g. reseting Tcl variables or object properties. Animations and slide-in effects can be defined: objects could slide in one by one from screen border (an effect well-known from PowerPoint).

The opposite of onEnter is onExit. This script will be executed after exiting a slide (meaning the slide becomes invisible). This script type is rarely used. One example is to evaluate the changes of a slide (input in text fields or an objects is moved by the student) or to check the answers of a quiz given on that slide.

The onRepeat script will be evaluated continuously again and again as long as a slide is visible. This is useful for complex animation steps in simulations. One can use this script to manipulate elements continuously. For example you can define a

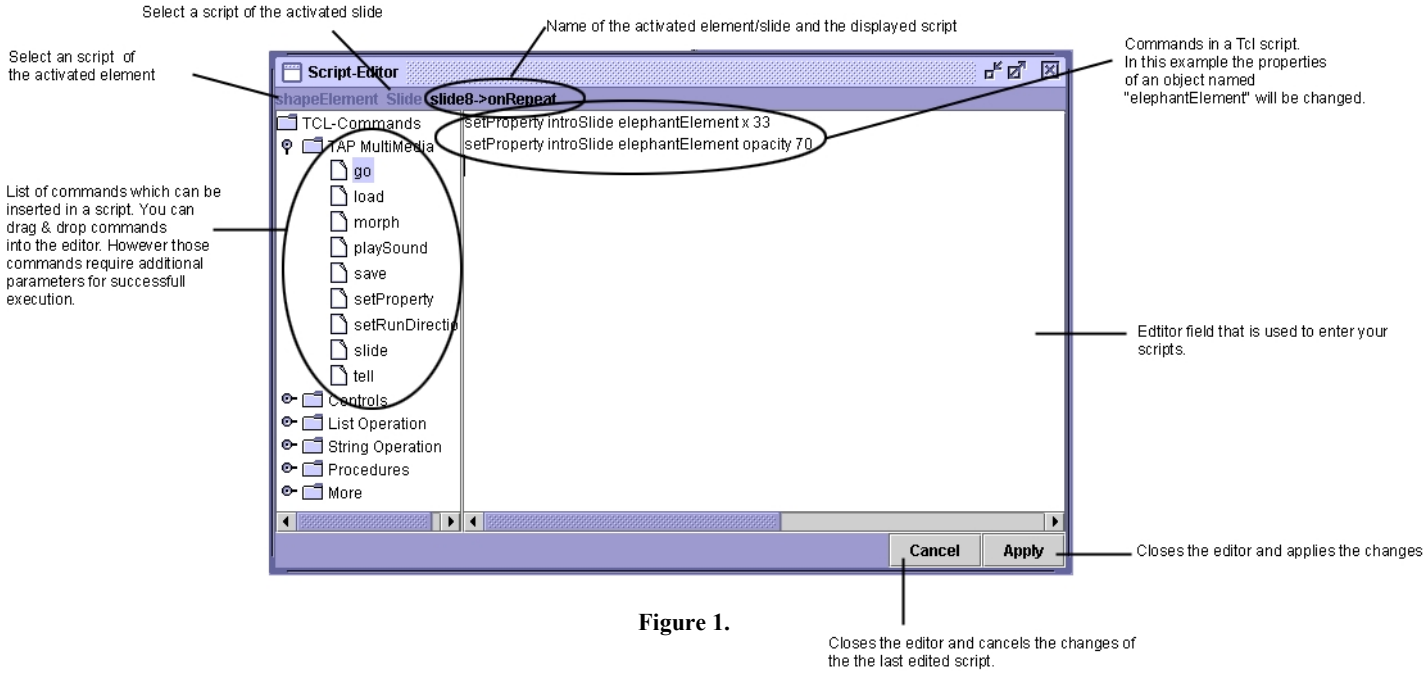

Figure 1.

command to move an element exactly five positions. Since the script will be executed repeatedly, the element will continue to move 5 positions again and again. Of course it is more interesting to calculate the element movement steps each time the script is executed. By doing so you can define element behaviors for simulations or virtual experiments.

2.3.2. Object scripts

The second category of Tcl scripts in Javanti are object scripts. Those scripts belong to one object instance. It is possible to define individual scripts for each object on the board. For each event that possibly occurs in the object there is a script property available. That script will be executed if the event actually occurs at runtime.

Objects support the regular GUI mouse and key events. Some objects have properties for uncommon event types: For example there is a quiz object. This object can be used to assign multiple choice tests. After the quiz object evaluated the user input it marks the result as correct of false. Here the quiz object considers the input of the correct answer as an event. A false answer would generate another event. For both events one can define a script.

From a technical point of view Javanti objects store Tcl scripts in simple string properties.

2.4. New Multimedia Commands

In this section we consider some of Javanti's command extensions to implement multimedia effects and object communication.

2.4.1. go command

To navigate to other slides of a Javanti course one can use the go command. Users can step forward and backward or address a slide by its name:

go next – presentation steps forward
go prev – presentation steps backward
go slide slideName – the slide „slideName" will be displayed

2.4.2. setProperty commmand

The command setProperty changes a property at course runtime, e.g. one can change the color, location or size.

The parameters of setProperty define:

- Which property to change (propName)
- The new property value (propValue)
- Which object to change (objName)
- Which slide contains the object (slideName)

The syntax of setProperty is this:

```
setProperty slideName objName propName propValue
```

Here is an example:

```
setProperty contentSlide exampleObj width 30
```

In the example we are addressing the element „exampleObj" which is part of „contentSlide". We change the width to 30% of the current window size.

2.4.3. getProperty command

To get the current value of a property we created the command getProperty. This command needs as parameters the name of a slide, the name of an object and the name of a property:

```
getProperty slideName objName propertyName
```

If you want to move an object relative to its current position, you can use the current property values. The next example moves an element 10 units to the right:

```
setProperty slideA elementB x \
    [ expr 10 + [getProperty slideA elementB x] ]
```

2.4.4. morph command

With setProperty one can directly set a property value. However in many cases a smooth change of the property values is a better approach. If you directly set the x location value from 10 to 50 the object jumps on the screen in one step. There is another command that lets you change a property's value in a certain number of steps. With each step the property value approaches the end value.

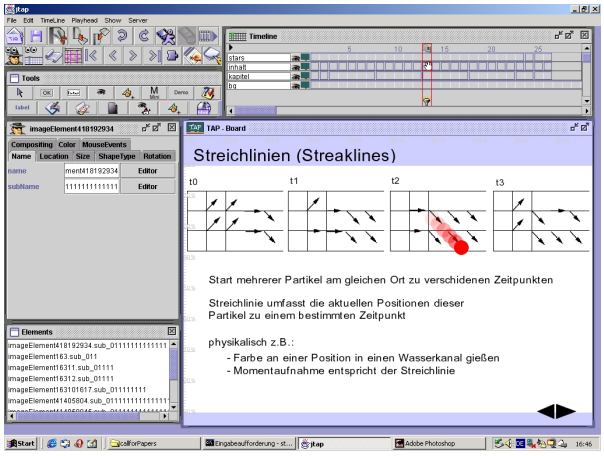

Figure 2.

The animation duration is defined in time steps. The syntax of the morph command is almost the same as in setProperty. Only the number of animation steps is appended as an additional parameter:

```
morph slideName objName propName propValue steps
```

In steps we define how many time steps should pass until the end value is reached. The property value of an element will continuously approach that value.

Here is an example:

```
morph animationSlide example x 50 10
```

The object „example“ is contained in the slide „animationSlide“. Starting from its current x location it will move to the new position x = 50. The animation will last for ten time steps. The object „example“ will be positioned at nine other locations before it finally reaches the end location.

In another example (Figure 2) we are using the morph command to animate a red particle from its original location to a new destination:

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

2.4.5. slide command

The slide command is similar to the morph command. However animation does not start with the current property value. The start value is an additional paramerter of this command. With slide you can animate properties from any start value to any end value. The syntax is this:

```
slide slideN objN prop startValue endValue steps
```

2.4.6. notify command

With notify one can send any type of messages to an object. The parameters include the name of the slide containing the object, the name of the object and the message:

```
notify slideName objectName „any message“
```

The way an object interprets the received message is up to the object. Objects of different types may interpret the same message in a different way.

A message can be both very simple or complex. Simple messages could be single words, whereas complex messages can contain instructions or even program code.

3. Create interactive eLearning materials

3.1 Applications of Tcl

Applications of Tcl in Javanti 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 kind 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. Conditional expressions can be used to select the next test page or a 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 time to the same page. Eventually Tcl scripts can be used to implement simulations or virtual experiments within an eLearning course.

You may find some of the following examples very simple. Of course we had to reduce the example code to a minimum. We are just providing some ideas how Tcl can be used in eLearning software. The simplicity also demonstrates the power of Tcl in combination with Java objects. With a few lines of code one can generate impressive results.

3.2. Tcl and Javanti commands

3.2.1. Data Output

You can use Javanti’s multimedia commands the same way as you use Tcl commands. You can mix both Tcl commands and Javanti actions in the same script.

Usually scripting languages such as Tcl get their user input from a text based console. However Javanti does not have any console. Rather Javanti provides a graphical environment. You can use all Javanti objects for user input.

To present messages or results of computing processes you can manipulate object properties. For example a text object on the board can display messages if you set its property „text“. Since you can use Javanti commands the same way like Tcl command, you can set Javanti properties within a Tcl script:

```
...
setProperty aSlide anObject text „Hi World“
...
```

Besides text based feedback, you can also manipulate colors, sizes, locations and any other properties within a Tcl script.

```
if {$location == „left“} {
  setProperty aSlide anObject x 0
} else {
  setProperty aSlide anObject x 80
}
```

In this example the value of the variable „position“ is compared to the string „left“. If the content matches, the object will be located at the left side of the screen (x=0). Otherwise the object will be located at the right side of the screen (x=80).

3.2.2 Reading property values

As we have seen the program output is shown to the user by manipulating the properties of objects. User input on the other hand can be read by getting the current property values of an object. The user of a course can manipulate some object property values directly on the graphical user interface:

- He can enter text into text fields. Thus, he changes the property „text".
- He can drag objects to other locations (if an object is dragable) and by doing so he changes the properties x, y
- Objects themselves can contain any type of graphic user interface components, such as input fields, buttons, active areas, etc. By activating or changing these components, the object may change some of its properties (depending on the object type).

To use property values as user input we can use the getProperty command. The next line assigns a property value to a Tcl variable:

```
set myVar [getProperty aSlide anObject y]
```

3.3. Use variables in courseware

3.3.1. Course data

Variables are used for many purposes. In the context of learning materials and courseware variables can be used to compute and store test results, generate user profiles and track user data. Variables can be used in conditions which will control the work flow of a learning session.

You can use variables to store user names:

```
set username [getProperty contentSlide user text]
set age [getProperty contentSlide age text]
set sexInput [getProperty contentSlide sex x]
if {$sexInput > 50} {
  set sex m
} else {
  set sex f
}
```

The user name and age can be read directly from two text fields. The sex of a user can be defined by dragging a graphical object to either the left or right side of the board. If the user drags the object to the right border (x > 50) he will be identified as a man (m). Otherwise she is considered as female (f).

You can use this user input anywhere in your learning application to make it more individual for a user.

```
setProperty content helloObject text \
  „Hello $username, here is your assignment:"
```

You can adapt the questions according to the user's age:

```
if {$age < 12} {
 notify quizSlide questionObject easy
} else {
 notify quizSlide questionObject difficult
}
```

Here we set the level of difficulty according to the age. If we are teaching a young student we send a message to the assignment object "questionObject" to switch to easy questions. Otherwise we notify the assignment object to ask difficult questions.

You can also adapt the visual appearance of slides taking the user settings into account:

```
if {$sex == m} {
  setProperty background bgObjects blue 255
} else {
  setProperty background bgObjects red 180
}
```

For man we use a blue background color, wheras woman will work with a cute red background.

3.3.2. Store test results in variables

You can use variables to count the number of correct and false answers a user has given:

```
set correctAnswers 0
set falseAnswers 0
set answerRatio 0
```

There is an example procedure in which we:

- increment the number of correct or false answers
- calculate the ratio between correct and false answers
- display the variable content in a text element (data output)

```
proc processAnswer {answer} {
  # access global counter variables:
  global correctAnswers
  global falseAnswers
  global answerRatio

  if {$answer == „correct"} {
     incr correctAnswers
  } else {
     incr falseAnwers
  }

#Calculate ratio between correct and false answers
#Multiplication by 1.0 forces Tcl to calc with
#real numbers

 set answerRatio \
     [expr correctAnswers * 1.0 / falseAnswers]

# output:
  setProperty background infoOutput text „Correct:
$correctAnswers --- False:\ $falseAnswers ---
Ratio: $answerRatio"

}
```

You can now invoke the procedure each time the user entered a correct or false answer. For example in Javanti there is quiz object supporting Tcl scripts which will be executed after the user entered a correct or false answer. There one could invoke the procedures:

```
# call for right answering
processAnswer correct

# call for false answering
processAnswer false
```

3.3.3. Conditioned Learning Flow

One could use answerRatio in conditions. For example you can deny access to certain slides if the user failed to succeed some previous tests. You could add a button to a slide which says „Next chapter“. In the mPressed action list of the button you first check the answer ratio:

```
if {$answerRatio >= 1 } {
  go slide chapter33
} else {
   setProperty background warningHint text „You\
can only start with the next chapter after you\
correctly answered at least 50 percent of the\
questions.“
}
```

If answerRatio is greater or equal to 1, the playhead jumps to slide „chapter33“. Otherwise (the ratio is smaller) a warning hint will be displayed. The warning hint is an object in the background and it was set invisible before.

3.4 User tracking

As we have learned you can use if-statements to control the content flow in a learning session. The simplest way is to deny or permit access to certain slides depending on former test results.

Using if-statements one can introduce a more granular control to courses. For example we could check the answerRatio after each question. If the value sinks under a minimum we could start an additional section for the student:

```
if {$answerRatio < 0.3} {
   go slide extraHelp
}
```

We can control the navigation flow by considering the history of displayed slides. We can use a Tcl list to store a list of slides the user has already used:

set listSlides ““

We would init the list usually on the first slide in the timeline. All slides of interest should add their names to the list when they are displayed. This can be done in the onEnter action list of the regarding slide:

onEnter for slide „intro“:
```
lappend listlides „intro“
```

onEnter for slide „physics“:
```
lappend listSlides „physics“
```

onEnter for slide „vectors“:
```
lappend listlides „vectors“
```

Now a jump to an additional slide „flowVisualization“ could depend on whether the slides „physics“ and „vectors“ have been displayed before:

```
if {[lsearch listSlides „physics“] != -1   \
    && [lsearch listSlides „vectors] !=-1 {

go slide „flowVisualization“

} else {

setProperty background warningHint text „Please \
read the slides ‚Physics‘ and ‚Vectors‘ first.“
}
```

3.5. Adaption of displayed information

So far we used conditions only for navigation. We can also use conditions to decide which elements are visible or not. For example if the answerRatio is on a low level, slides could contain extra information and explanations. We could even show or hide navigation objects to only enable navigation options that are permitted. We can show or hide complete text blocks or any graphics depending on condition statements.

The following examples can be placed in onEnter scripts to define the visible content before the slide is put on the overhead projector:

```
if  {$answerRatio < 0.5} {
  setProperty background helpButton visible true
} else {
  setProperty background helpButton visible false
}

if {$answerRatio >= 1.0} {
  setProperty background nextChapter visible true
}
if {$pageCounter >=3} {
  setProperty contentSlide extraInfo visible true
}
```

3.6 Regular expressions

Tcl supports regular expressions. This is useful to check the input of answer fields.

Using regular expressions we can allow error tolerance for the user’s answer. We can use one regular expression to accept all of the following variations of the word „Vancouver“:
„Vancouver“, „Vaaancoouver“, „vancouver“ etc.

3.7. Create 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 behavior 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 Javanti.

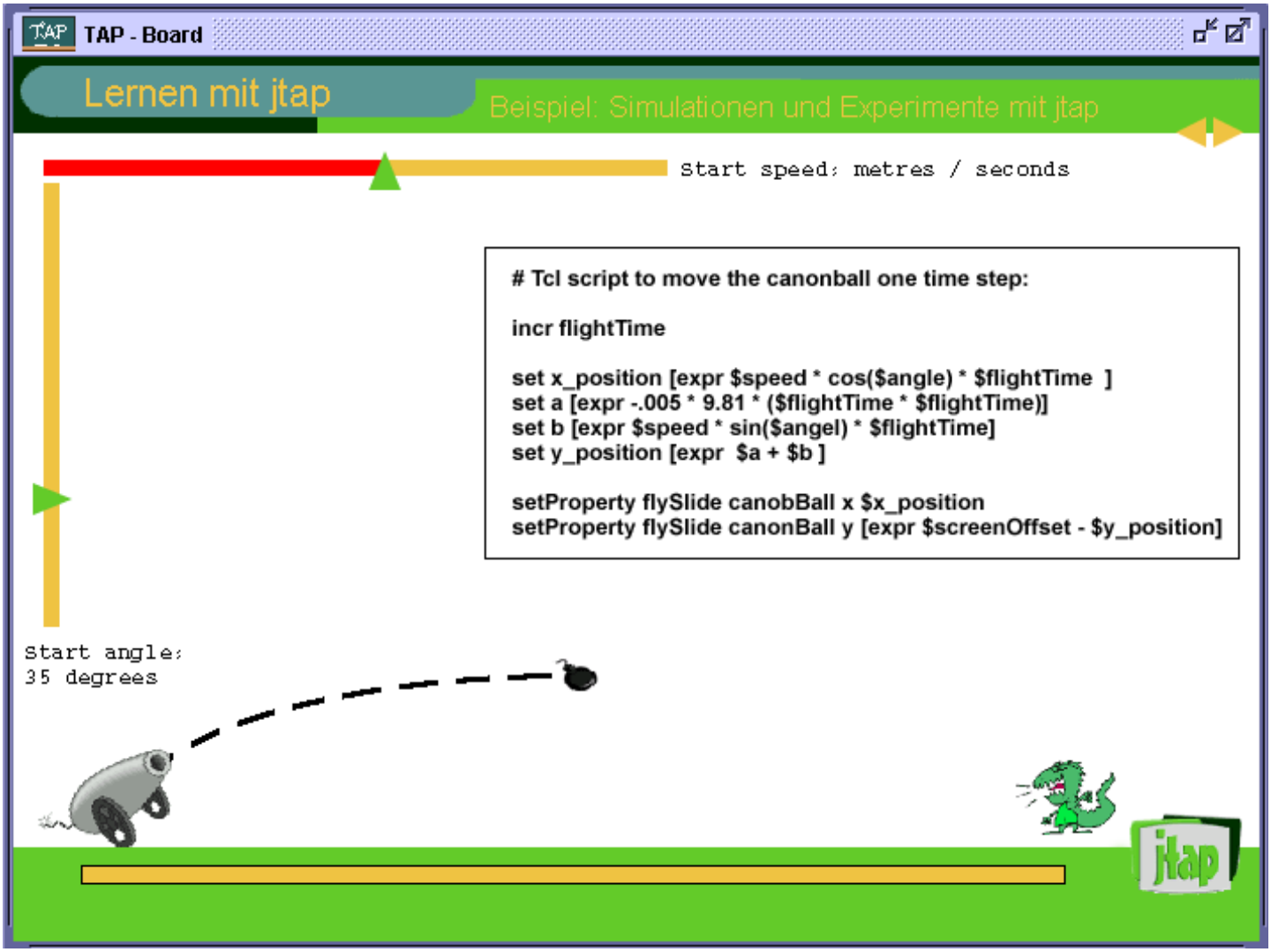

Figure 3.

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 continiuous 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 continiuously as long as the slide is showed, 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 to drag other objects into the gravity field or change the gravity constant. Finally in the onExit script we can analyze the current position of the objects and store the results in variables to access those within other slides.

In the example shown in Figure 3 the onRepeat script defines the canonball'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 canon.

4. Conclusion

Using Tcl in our project guaranteed the use of a well-developed language. The easy-to-use syntax allows every newbie to write simple actions, while the more experienced developer is satisfied, too. Javanti's event-driven use of Tcl scripts does not limit the use of Javanti for eLearning applications but provides a way to build graphic user interfaces for Tcl in general. Also the use of Tcl and Java in combination unites the advantages of both worlds.

5. About the project

Javanti 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. Several thesis have discussed design issues of the software. Christian Kohls and Tobias Windbrake who originally developed the ideas for this software are going to start up a company. They offer professional consulting, course development and software engineering to the educational market. They have successfully used Javanti in commercial projects. The software itself is an Open Source product and available at the official web page: http://www.Javanti.org

6. References

http://www.jtap.org
http://www.tcl.tk/software/java/
http://www.t-ide.com/tcl2002e.html

“eLearning Development Environment JTAP – New Approaches to eLearning Content Creation” / Proceedings of the Second Research Workshop of EDEN – Hildesheim, Germany 2002

“Multimediales Self-Authoring-Tool jtap im kooperativen Arbeitsmodus” / Workshop "CSCL - Kooperatives E-Learning", GI-Jahrestagung Informatik 2002; http://ipsi.fhg.de/concert/cscl-02/index.html