SophieScript - Active Content in Multimedia Documents

Jens Lincke[1] Robert Hirschfeld[1] Michael Rüger[2] Maic Masuch[3]

[1] Hasso-Plattner-Institut, University of Potsdam
{jens.lincke, hirschfeld}@hpi.uni-potsdam.de

[2] Impara GmbH, Magdeburg
michael.rueger@impara.de

[3] University of Applied Sciences, Trier
masuch@fh-trier.de

Abstract

Active content in multimedia documents helps the reader to grasp the implications of nonlinear and complex systems that are difficult to understand in a text-based description. The readers are able to make their own experiments by changing the underlying rules of these systems. The multimedia authoring environment Sophie has limited capabilities for authoring active dynamic contents. For that reason we integrated a tile scripting system based on Tweak Etoys in Sophie. In SophieScript scripts can be embedded into the text and the reader changes them via direct graphical manipulation. With the implemented tile scripting system the user can easily create dynamic content and, at the same time, make use of the multimedia and text capabilities of Sophie.

1 Introduction

Standard multimedia documents are typically composed of text, images, audio, and video. Scripting languages are one of many paradigms to create (inter-)activity in these documents by synchronizing, animating, and connecting existent contents. Scripting languages can further be used to create interactive content simulations or games. When scripts are used this way, they are part of the content, but in an invisible way; only their effects are visible to the reader of the documents. In Active Essays [7] the scripts themselves become the content. They are shown to the user, they can be changed, and they can change other contents. So, scripts become an active content that enables the users to make their own experiments by not only changing some parameters, but by changing the underlying code of the dynamic systems (e.g., like simulations of epidemics [14]). By playing and experimenting with scripts users have an immersive way to explore a given environment. We want to bring that kind of active content to the authoring environment Sophie [21].

Sophie combines multimedia content with advanced text processing features, but it has limited capabilities for authoring dynamic content. Dynamic contents in Sophie is created by Timelines, Triggers, and Actions. These tools enable the user to create simple dynamic behavior in an easy but restricted way. Actions (like switching to a page or displaying a frame) are predefined and can be triggered by events (like a mouse click). A scripting language or system would widen the range of possible dynamic content by allowing more complex operations like control structures and variables. The problem is that textual scripting languages (like JavaScript) are not suitable for Sophie's target users, because Sophie defines itself as a tool to make the creation of multimedia books possible without programming.

A solution to create complex dynamic content without using a programming language is Etoys [9]. Etoys is a constructivist tool for children and uses visual programming to bring life into a world with objects by creating scripts out of graphical building blocks called tiles.

Scripting in other multimedia documents like Flash [2] or HTML documents distinguishes between author and reader mode. This means that scripts are created by an author and can change, generate, and present content at runtime. This is not sufficient for media formats like Active Essays [7]. In Active Essays the reader should be able to experiment with the rules of dynamic systems by changing them to get a better and deeper understanding. This idea is based on constructivist theories by Piaget and Papert [16]. To support this, a scripting environment for active content has to enable the reader of a document to change scripts in a user-friendly way.

Readers of standard multimedia documents only see the effects of scripts, but cannot change them. Active content demands that the user has full freedom of changing the dynamic content and should not be restricted to alter some parameters (like configuring a simulation). The user should be able to experiment with the whole script and change it to understand and learn from it. To make this possible, our system not only allows the manipulation of content through scripting, but scripts themselves become content and are changeable at runtime by the reader.

The reading of documents with active contents should not be limited to people who are able to read and write in a specific programming language. The use of tile scripting allows the reader to experiment with scripts without needing to know a programming language and be familiar with its syntax. This is why we used a visual programming language like Etoys: the main advantage of tile scripting is the elimination of syntax errors. To lower the barrier of programming for the user we choose Tweak Etoys as a scripting system, which is also used in TinLizzie [15].

The remainder of the paper is organized as follows: Section 2 gives a short overview of Sophie, Tweak Etoys, and our integration of these two systems: SophieScript. Scripting of active content is divided into two main aspects: The scripting of Content is described in Section 3 and scripts as content are covered in Section 4. Section 5 shows an example application of active content, in which a turtle draws a tree. Section 6 discusses related work. The last Section draws a summary and gives an outlook on future work.

2 Integration of Sophie and Etoys

SophieScript is based on two systems: it integrates the tile scripting from Tweak Etoys into the Sophie multimedia authoring environment. Both systems are implemented in Squeak [4].

2.1 Sophie

Sophie [21] is an all-purpose tool for dealing with media. It allows users to easily create books that contain any sort of media on hand – text, images, sounds, videos, and animations. Sophie does for media what a physical book does for text and images: With Sophie, authors can create multimedia books. You might think of it as a wrapper for anything digital, but it is more than that. Sophie differs from previous platforms for electronic reading by giving the author as much control over the form as over the content.

Sophie is media-agnostic: All media is the same inside of Sophie. You can create a book based on a long piece of text, like a traditional novel. Or you can create a book based on a series of photographs, something like a slideshow, adding narration or a soundtrack to play with the rhythm. Sophie aims at having a low enough threshold for non-professional users to create media-rich dynamic contents without resorting to programming. Sophie's current versions are essentially closed applications, whereas Etoys tries to open up the whole system to the user for exploration.

2.2 Tweak Etoys Tile Scripting

The Tweak Etoys scripting system is a visual programming with graphical building blocks called tiles. It is implemented in Tweak [17] and was planned as a successor of Squeak Etoys [9].

We use Tweak Etoys as a scripting language, because it allows the user friendly creation and editing of scripts with the elimination of syntax errors. This is necessary, because scripts are shown to and edited by the author but also by the reader of the documents, who must be able to change the scripts without knowing the scripting language.

Tweak Etoys integrates itself deep into the underlying system and allows the creation of scripts, which get compiled as normal smalltalk methods. It supports variables, conditionals, loops, recursion, and can use the full event architecture of Squeak.

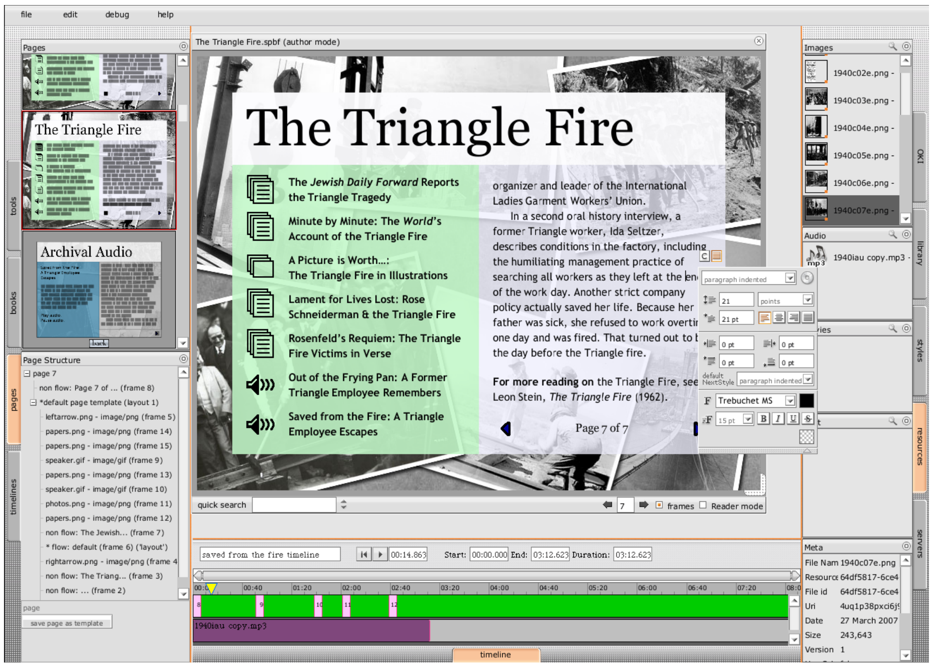
These features make it more powerful than Squeak Etoys and Scratch [13], but at the same time more difficult to handle. It is no full featured scripting language, because it misses features like local variables, return statements, and arguments for self-defined scripts. But these features could be added later. The user-friendliness of the scripting through drag and drop is more important than using a feature complete scripting language that the user is not able to read and write and with which he can not safely experiment.

2.3 SophieScript

SophieScript is the prototypical integration of the tile scripting system of Tweak Etoys into Sophie. The system allows the scripting of Sophie's content and makes scripts themselves content. These scripts are unlike other scripts in multimedia documents visible to the reader and can be changed at runtime by the reader. The first aspect of SophieScript is the scripting of content, that means that the scripts provide dynamic behavior for content in Sophie through the use of an indirect layer of scripting objects called Scriptees. The second aspect is that scripts are presented to the reader, are part of the text, and can be changed by the reader through direct graphical manipulation in the typical Etoys manner.

3 Scripting of Content

Finding a suitable model of scripting content in Sophie is difficult, because it was created as a productive tool and not like Etoys with a simple model for scripting in mind. Squeak Etoys scripts only one kind of object: the Player; and uses all kinds of Morphs [12] as costumes for that


The Triangle Fire.spbf (author mode)

The Triangle Fire

The *Jewish Daily Forward* Reports the Triangle Tragedy

Minute by Minute: The *World*'s Account of the Triangle Fire

A Picture is Worth...: The Triangle Fire in Illustrations

Lament for Lives Lost: Rose Schneiderman & the Triangle Fire

Rosenfeld's Requiem: The Triangle Fire Victims in Verse

Out of the Frying Pan: A Former Triangle Employee Remembers

Saved from the Fire: A Triangle Employee Escapes

organizer and leader of the International Ladies Garment Workers' Union.

In a second oral history interview, a former Triangle worker, Ida Seltzer, describes conditions in the factory, including the humiliating management practice of searching all workers as they left at the en of the work day. Another strict company policy actually saved her life. Because her father was sick, she refused to work overti one day and was fired. That turned out to b the day before the Triangle fire.

For more reading on the Triangle Fire, see Leon Stein, *The Triangle Fire* (1962).

Page 7 of 7

saved from the fire timeline 00:14.863 Start: 00:00.000 End: 03:12.623 Duration: 03:12.623

Figure 1. Screenshot of Sophie

player. The structure and application programming interface (API) of objects in Sophie are not suited for scripting them directly, because they are designed to only work well with tools but not stand alone. As a solution to this problem, we introduce scriptees as an indirect scripting layer.

3.1 Model of Content in Sophie

Books in Sophie consist of pages with text, images, audio, and video. An example book opened in Sophie is shown in Figure 1. The text of a book flows into series of frames that can be distributed on a spine of pages. This allows multicolumn and other forms of layout as well as automatic distribution of long contents on many pages. This distinguishes Sophie from other multimedia authoring systems that put graphics into their main focus and only allow text in non-flowing frames. Images and videos are also positioned with frames, which makes them the primary graphical object in Sophie. Text is represented as nodes in a content tree and all contents can be automatically formatted by using styles.

3.2 Defining a Scripting API

There is no API to directly interact with Sophie objects, because most operations need other objects like an editor or importer that do the actual work. This ensures that every change of state goes through the application, so that plugins and tools can react on it.

To define a secure and user-friendly scripting API, we use special objects for scripting called "scriptees" as a layer of abstraction. These scriptees provide a scripting API and hide the system API of the objects from the user. The objects themselves could implement an API for scripting as long as there are no naming conflicts, but there is no standardized way to separate and secure method protocols in Squeak.

Scriptees resemble players in Etoys, the main difference is that they are no graphical objects and simply provide the infrastructure for scripting. Players stand on their own in contrast to scriptees, which wrap around other objects. They are only helper objects for them. For efficiency, the objects in Sophie that represent the model are no Tweak objects. Tweak EToys can only work with Tweak objects, because they provide the basic scripting functionality. This is the technical reason for separating the scripting aspects of those objects into scriptees.

3.3 Infrastructure for Scripting

Scriptees should be Sophie objects and Tweak objects at the same time. Because there is no multiple inheritance in Squeak, the SophieScriptee class is subclassed from CObject and implements the protocol of the class SophieObject so that it can live in both worlds. This enables scriptees to use the asynchronous event system and allows them to be scripted and, at the same time, integrated into the serialization framework of Sophie. Like in Squeak Etoys the

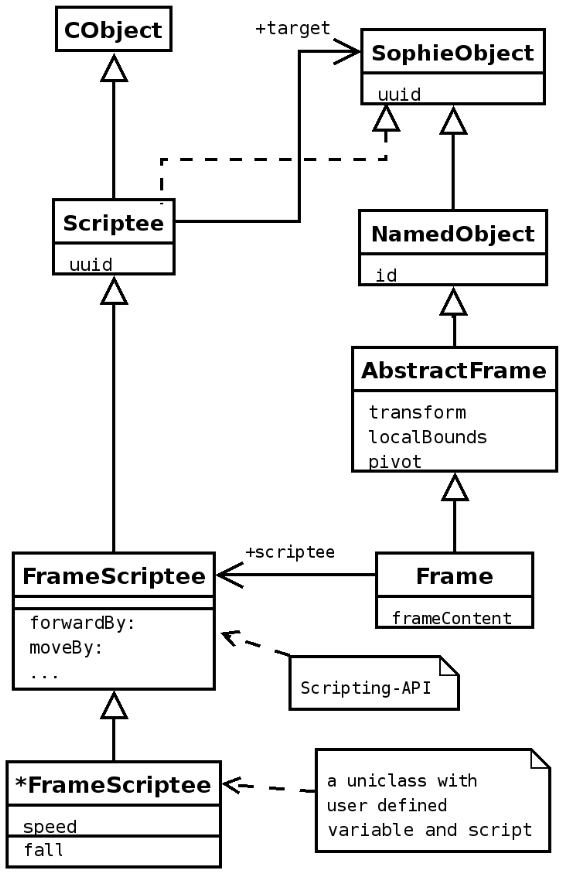


Figure 2. Relation of Scriptees to Sophie Objects

scripts themselves are compiled as methods to uni-classes, which are anonymous classes that enable instance specific behavior, so that each scriptee can have its own scripts (see Figure 2).

The scriptee delegates all state changes to its target object and retrieves states from its target through the use of Tweak's virtual fields. These virtual fields cannot automatically generate the necessary change events themselves, because they cannot look into normal Sophie objects, which are no Tweak objects. The translation of state changes in Sophie to events for scripting is needed to update the values in the viewers and inspectors of scriptees. This problem can only be tackled individually, because there is no automatic generation of events. When variables change in normal Squeak objects, these events have to be manually signaled in the appropriate places (like editors). The implementation for the class FrameScriptee uses the existing events in the user interface to work around that problem, but this is no general solution for other Sophie objects. A general but not so efficient solution would be to generate the change events through polling mechanisms.

3.4 Integration with existing Sophie Tools

Sophie provides predefined actions like “Go To Frame” or “Toggle Page“, but the users cannot define their own actions. SophieScript adds ScriptActions to connect the trigger with Tile-Scripts.

TileScripts can be Sophie content in two different ways. They can be embedded into text or act directly as frame content. When they are frame content, they behave like in Etoys. That means they are arranged by the user and behave strictly like graphical objects. When the script is part of the content tree, they flow with the text like embedded images.

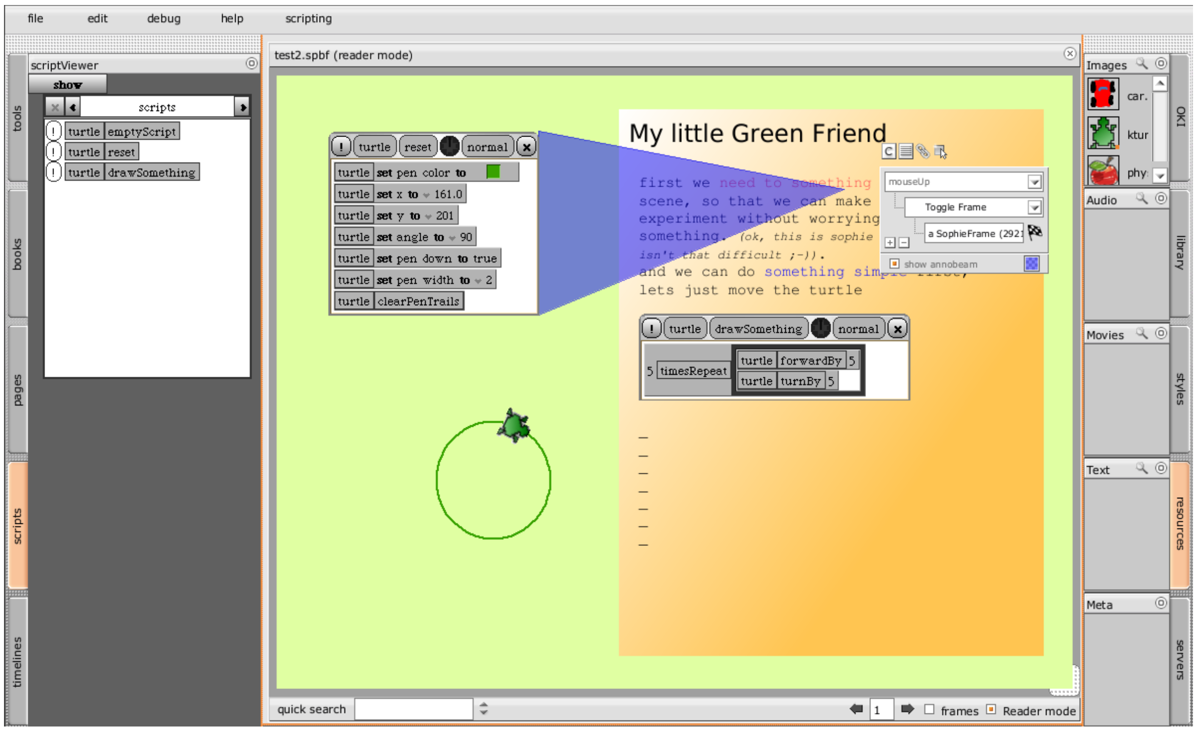
The scripting user interface, as shown in Figure 3, integrates the Etoys viewer into the Sophie docs. The halo of Etoys is replaced in Sophie by a HUD (Head Up Display) and handles. The HUD provides editors for styles and other object related content directly at the object and not in a tool bar. The handles position and resize the frame. The proper use of these HUD and handles in SophieScript should be addressed in future work.

4 Scripts as Content

Scripts in Active Essays are not only capable of modifying and creating other content, but are also content themselves. These scripts are part of the text and the reader can edit and run them for experimentation. Thus, in SophieScript the scripts become active contents.

4.1 Tile Nodes

To unify scripts and text in Sophie, we decided to represent scripts as content nodes of the book. This is analog to the decision to represent scripts with tile players in Tweak Etoys, because players are the main building material in Tweak and content nodes are the building material of text in Sophie. Thus, the scripts are represented as tile nodes to unify the handling of text and scripts. Tile nodes are book content nodes, which are used in Sophie to represent structured text. This is different to scripting content in HTML, where the dynamic behavior is represented in a different language as the content itself. The representation of scripts as content nodes allows the seamless integration of scripts into the storage management with mechanisms like copy and paste. Another advantage is that the nodes make the script independent of a parser and compiler of a specific programming language. It is conceivable that they can be transformed and adapted to platforms like web browsers, without the need to implement a full Smalltalk environment there.



```
turtle emptyScript
turtle reset
turtle drawSomething
```

```
turtle reset normal
turtle set pen color to
turtle set x to 161.0
turtle set y to 201
turtle set angle to 90
turtle set pen down to true
turtle set pen width to 2
turtle clearPenTrails
```

My little Green Friend

first we need to something scene, so that we can make experiment without worrying something. *(ok, this is sophie isn't that difficult ;-)).* and we can do something simple first, lets just move the turtle

```
turtle drawSomething normal
5 timesRepeat
    turtle forwardBy 5
    turtle turnBy 5
```

Figure 3. The Scripting User Interface

The representation of scripts as nodes leads to a separation of the user interface from its persistent model. In text-based scripting languages this is the source code. This separation allows the use of a graphical user interface (GUI), like the drag and drop behavior of tile scripting, to edit scripts.

4.2 User Interface

We experimented with the Sophie text compositing to display the scripts and to integrate the rendering of scripts into the Sophie architecture. This would have enabled the use of high-quality font compositing and rendering. But the results where not satisfying, because the architecture of editing and compositing text was too different from the graphical nature and the drag and drop user interaction of tile scripts.

The content node structure itself is not suitable for implementing drag and drop, because it is only loosely coupled with the graphical structure. To change this, the graphical features of Tweak would have to be reimplemented into Sophie. Since this is not practical, we use the existing Tweak tile player and costumes for the user interaction. This creates a need to synchronize the player structure with the tile node structure, which represent the scripts. For the synchronization of the node with the player hierarchy (see Figure 4), the composing and event mechanisms in Tweak could be utilized. So, a tree of tile players is generated and updated from the tile nodes, and when the user changes the graphical structure of the players, the nodes get updated.

4.3 Building Blocks of Scripts

Beside nodes and players, there are other objects that represent the building blocks of scripts at a time. These semantic elements of scripts are represented by objects of four different class hierarchies:

1. BookContentTileNodes provide the elments of the model of the script in Sophie and are used for serialisation of scripts.

2. CTilePlayers are responsible for user interaction, for example the application of the types of a tile to ensure that scripts can only be edited in syntactically correct ways.

3. CTileCostumes display the elements of the script, they can be seen as a view on the model.

4. CParseNodes are used by the CTilePlayers building a parse tree, which gets compiled into a Squeak Method.

The hierarchies provide the elements of trees, which are transformed into each other. Since the requirements of the different contexts differ, there is only a loose mapping of

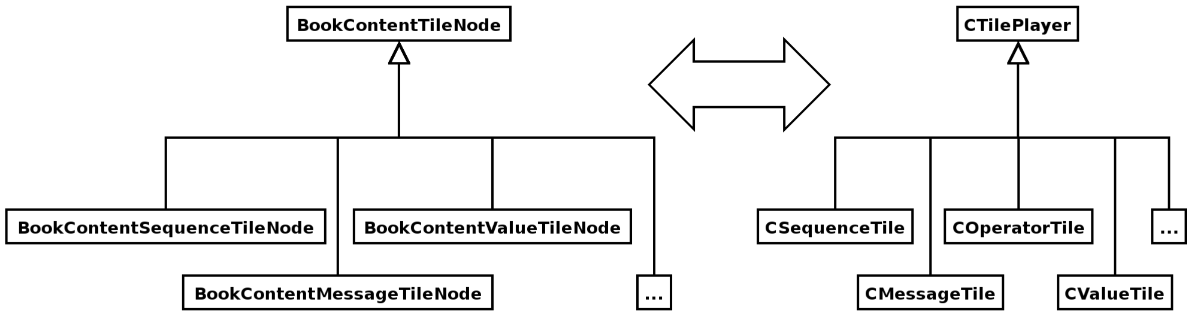


Figure 4. Bidirectional Mapping of the TileNode Class Hierarchy and TilePlayer Class Hierarchy

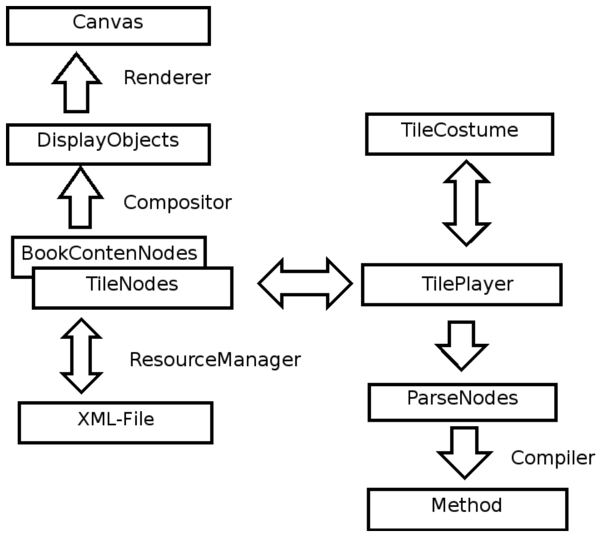


Figure 5. Dataflow of a Script

elements. The costume tree has much more elements than the player tree, because the costumes contain graphical elements (e.g. buttons and other widgets), which have no correspondence in the model of the script. The conversion between these representations for a script are shown in Figure 5.

5 Example: Drawing a Tree

To illustrate the possibilities and limits of the implemented scripting system, we choose turtle graphics as an example for active content. Turtle graphics were a part of Logo [16] and are also part of Etoys.

An example of active content in a book is shown in Figure 6. The script in this example recursively draws a tree and shows that the script, the object and the final result are equally valid content for the user. The user can make experiments by playing with the script to gain an understanding of concepts like recursion, and to produce interesting graphics.

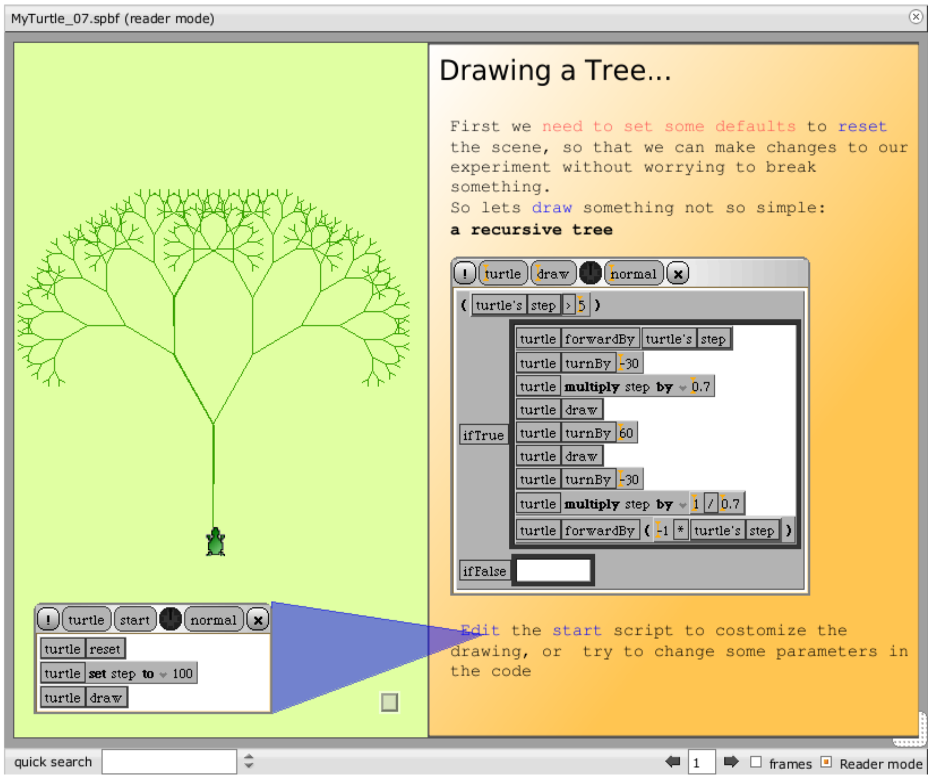
The turtle in Figure 6 is an image which is scripted to recursively draw a tree. The scripts can be started by clicking on marked text.

All scripts in these example belong to the turtle frame. They provide default settings, and thus the user can experiment and go back to a defined state. This has to be done by the author, because there is no distinction between the author and reader mode in the scripting environment. Since such maintenance scripts are not as interesting to the reader as the script with the tree drawing algorithm, they are hidden by default and can be displayed by clicking on marked text. To execute the scripts, other marked text can be clicked. This is an application of ScriptActions: the script is executed when a mouse down event is triggered on the marker.

6 Related Work

The first Active Essays from Alan Kay and Ted Kaehler covered evolution [7, 6] and allowed the experimentation with evolutionary algorithms. They used a web browser for the textual and graphical content and contained separate files for execution in Glyphic Script [11] and Hypertalk [22]. Later, the evolution essay was redone as an Etoys Project in Squeak [8]. Other authors experimented with Active Essays, too: Resnick and Silverman made an essay about exploring emergence [18] using a web browser and Java applets; Guzdial and Greenlee made a series of essays about music [3] using Squeak.

The TinLizzie WysiWiki [15] uses the Tweak Etoys as scripting system in a collaborative multimedia and scripting environment. The multimedia content is stored in the Open



MyTurtle_07.spbf (reader mode)

Drawing a Tree...

First we need to set some defaults to reset the scene, so that we can make changes to our experiment without worrying to break something.
So lets draw something not so simple:
a recursive tree

```
turtle draw normal
( turtle's step > 5 )
ifTrue
    turtle forwardBy turtle's step
    turtle turnBy -30
    turtle multiply step by 0.7
    turtle draw
    turtle turnBy 60
    turtle draw
    turtle turnBy -30
    turtle multiply step by 1 / 0.7
    turtle forwardBy ( -1 * turtle's step )
ifFalse
```

```
turtle start normal
turtle reset
turtle set step to 100
turtle draw
```

Edit the start script to costomize the drawing, or try to change some parameters in the code

Figure 6. A script drawing a tree

Document Presentation (ODP) format and the scripts are stored as Smalltalk source code in attributes of elements. OpenOffice preserves these attributes, so that the document can be edited without losing the scripts added by TinLizzie.

In books and other texts, programs are printed as listings of code with a special format. In HTML the pre-tags are often used to insert code that the user should read. Scripts that should be executed by the browser to change the content or its presentation, are inserted or referenced in script-tags.

Documents that demonstrate the effects of JavaScript have to insert the code into the document two times: first, as human readable text and second, as the real script. This allows the user to read and execute the script, but not to change it. A way to solve this issue is to create an interpreter in JavaScript for a language like shown in LogoWiki [1] or to use the `eval` function of JavaScript to execute JavaScript code at runtime as used in a JavaScript shell [19].

LogoWiki implements a Logo interpreter in JavaScript and integrates it into a simple JavaScript-based wiki [10]. Thus, active content is possible in standard environments like a web browser. The Lively Kernel [5] shows that Morphic like environments can also be implemented with JavaScript, so Etoys in a web browser is not that far away any more.

7 Summary and Outlook

To enable authors to create Active Essays in Sophie, SophieScript allows the integrated creation of active content with direct graphical user interaction. Scripts can change content and, at the same time, be changeable by the users themselves.

The objects in Sophie are indirectly scripted via scriptees, to provide a clean and secure scripting API.

The scripts can be content of frames or flow with the text, when they are inserted in the content tree.

They are represented by content nodes, so that they can be integrated into the Sophie architecture like the storage mechanism.

The Etoys tile scripting can benefit from adding features like local variables, arguments for scripts, return values, and collections to close the gap to textual scripting languages. These concepts are addressed by the architecture, but missing from the user interface, because they are in conflict with the global drag and drop of tiles. Their integration would allow to convert the textual script and graphical tile script, back and forth.

A bidirectional conversion between textual and graphical scripts (e.g., like Universal Tiles in the Etoys Evolution Essay [8]) would enable the advanced user to author quickly and freely, without losing the user-friendliness of the graphical interaction when needed.

Etoys is a prototype-based language [20]. Scripts define behavior directly for objects and behavior can be generalized through parent-child relationships. To adapt this for SophieScript, the existing inheritance relations of objects in Sophie, like template and style hierarchies can be used. An alternative is to model this prototype relation explicitly, as done in Squeak Etoys.

At present there is only a simple scripting API implemented for Frames. There are other classes and some features of the application itself which would need some scripting API. The integration of the application commands into the tile scripting system would allow the automation of the application.

The user interface for scripting is only a placeholder, it does not use the full possibilities of Sophie's HUDs and flaps.

There is a performance impact from scripts that change the model of the book in such a way that the text has to be reflown. In the worst case this can lead to the invalidity of every page in the book and this can take some time. The solution to restrict the scripting on display pages would solve this problem but only at the cost of express-ability.

SophieScript shows that active content can be a valuable component of Sophie to enable the integrated creation of Active Essays.

8 Acknowledgements

We thank Bert Freudenberg for sharing his invaluable insights into Tweak and Etoys, Bernd Eckardt for his fruitful discussions on scripting in Sophie, and Philipp Engelhard and Stefanie Quade for their comments on an early draft.

References

[1] Logo wiki. as of Oct 06, 2006, retrieved `http://www.logowiki.net` via `http://www.archive.org/`.

[2] J. Allaire. Macromedia Flash MX—A next-generation rich client. March 2002.

[3] M. Guzdial and J. Greenlee. A Computer Music Implementation Course Using Active Essays. 2002.

[4] D. Ingalls, T. Kaehler, J. Maloney, Scott, and W. A. Kay. Back to the future: the story of Squeak, a practical Smalltalk written in itself. *ACM SIGPLAN Notices*, 32(10):318–326, 1997.

[5] D. Ingalls, T. Mikkonen, K. Palacz, and A. Taivalsaari. Sun labs lively kernel. as of Oct 12, 2007, `http://research.sun.com/projects/lively/`.

[6] T. Kaehler. Evolution Part II. as of Aug 8, 1996, retrieved `http://www.research.apple.com/research/proj/learning_concepts/evolution_ii/evolution_ii.html` via `http://web.archive.org`.

[7] T. Kaehler and A. Kay. Evolution. Version 4.4, Nov 22, 1995, retrieved `http://www.research.apple.com/research/proj/learning_concepts/evolution_active_essay/evolution.html` via `http://web.archive.org`.

[8] A. Kay. Active essay about evolution. as of Aug 06 2007, `http://www.squeakland.org/whatis/a_essays.html`.

[9] A. Kay. Squeak etoys authoring and media. as of Aug 01, 2005, `http://www.squeakland.org/pdf/etoys_n_authoring.pdf`.

[10] A. Kay. A "little demo". email to olpc-software mailing list, Apr 10, 2006, `http://www.redhat.com/archives/olpc-software/2006-April/msg00035.html`.

[11] M. Lentczner. Glyphic script. In *OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications* [20], pages 104–106.

[12] J. Maloney. An introduction to morphic: The squeak user interface framework. *Squeak: OpenPersonal Computing and Multimedia*, 2001.

[13] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick. Scratch: a sneak preview [education]. *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*, pages 104–109, 2004.

[14] Y. Ohshima. Kedama: A GUI-Based Interactive Massively Parallel Particle Programming System. *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 91–98, 2005.

[15] Y. Ohshima, T. Yamamiya, S. Wallace, and A. Raab. Tinlizzie wysiwiki and wikiphone: Alternative approaches to asynchronous and synchronous collaboration on the web. In *C5 '07: Proceedings of the Fifth International Conference on Creating, Connecting and Collaborating through Computing*, pages 36–46, Washington, DC, USA, 2007. IEEE Computer Society.

[16] S. Papert. *Mindstorms: children, computers, and powerful ideas*. Basic Books, 1980.

[17] A. Raab. Tweak project wiki, 2005. as of Juni 26, 2007, `http://www.tweakproject.org`.

[18] M. Resnick and B. Silverman. Exploring emergence. as of 04 Feb, 1996, `http://llk.media.mit.edu/projects/emergence/`.

[19] J. Ruderman and T. Mielczarek. Javascript shell. as of Oct 30, 2005, `https://www.squarefree.com/shell/`.

[20] R. B. Smith. Prototype-based languages (panel): object lessons from class-free programming. In *OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications*, pages 102–112, New York, NY, USA, 1994. ACM Press.

[21] B. Stein and D. Visel. Sophie homepage. as of Aug 06, 2007, , `http://www.sophieproject.org/`.

[22] K. Wheeler. Hypertalk: The language for the rest of us, 2004.