# The Fábulas Model for Authoring Web-based Children's eBooks

Hedvan Fernandes Pinto
Federal University of Maranhão
São Luís, MA, Brazil
hedvan@laws.deinf.ufma.br

Carlos de Salles Soares Neto
Federal University of Maranhão
São Luís, Brazil
csalles@deinf.ufma.br

Sérgio Colcher
PUC-Rio
Rio de Janeiro, Brazil
colcher@inf.puc-rio.br

Roberto Gerson de Albuquerque Azevedo
PUC-Rio
Rio de Janeiro, Brazil
razevedo@inf.puc-rio.br

## ABSTRACT

Nowadays, tablets and smartphones are commonly used by children for both entertainment and education purposes. In special, interactive multimedia eBooks running on those devices allow a richer experience when compared to traditional text-only books, being potentially more engaging and entertaining to readers. However, to explore the most exciting features in these environments, authors are currently left alone in the sense that there is no high level (less technical) support, and these features are usually accessible only through programming or some other technical skill. In this work, we aim at extracting the main features on enhanced children's eBooks and propose a model, named Fábulas—the Portuguese word for *fables*—that allows authors to create interactive multimedia children's eBooks declaratively. The model was conceived by taking, as a starting point, a systematic analysis of the common concepts, with the focus on identifying and categorizing recurring characteristics and pointing out functional and non-functional requirements that establish a strong orientation towards the set of desirable abstractions of an underlying model. Moreover, the paper presents a case study for the implementation of Fábulas on the Web, and discusses the authoring of a complete interactive story over it.

## CCS CONCEPTS

• **Software and its engineering → Domain specific languages**; Model-driven software engineering; • **Applied computing** → *Hypertext / hypermedia creation*;

## KEYWORDS

Interactive eBook; Multimedia Authoring; Conceptual Model

## 1 INTRODUCTION

With the advent of the Web and the popularization of portable electronic devices such as tablets and smartphones, electronic books (eBooks) have found a fertile environment to proliferate. Indeed, since the creation of the first eBook in 1971 by Michael Hart [15], eBooks have evolved to become not only a copy of printed books but also to include many new features available in the digital world, such as multimedia content and interactivity [4]. In particular, picture eBooks for children [30] extensively take advantage of those enhancement features to engage and entertain children on stories while serving as an educational platform as well.

Currently, to deploy enhanced eBooks for children, authors can use one of the format standards for eBooks, such as PDF [17], ePub [9], and Mobi [2], each one with their advantages and disadvantages. PDF, for instance, was created to provide a high-fidelity layout, similar to printed publications, whereas ePub and MOBI allow the adaptability of the layout for different screen sizes and devices. At some extent, those formats can also be supported in modern web browsers through plugins or using a polyfill approach [8] with HTML5/JavaScript. Another currently widely used approach for developing and delivering interactive eBooks is through software (or *apps*) in proprietary stores (e.g. Google Play and Apple Store) in which case they are sometimes called *book apps* [27, 30].

Some of the current eBooks formats can contain interactive elements. For instance, PDF supports hyperlinks and animation using videos or Flash (.SWF) [1]. Even so, it cannot fully support all the features required for elaborated interactive children's eBooks, such as audio recording and keeping control of complex object states during the story path. Other formats, such as ePub3 or native book apps, when taking advantage of imperative programming languages and low-level device APIs, can indeed support all the usual features on enhanced children's eBooks. However, for non-trivial features, such as animations and controlling object states during the story, these latter approaches do not provide high-level abstractions and require the authors to have technical skills.

Independently on how their final content is deployed, interactive children's eBooks usually share a common structure and use similar features. In this work, we aim at extracting those features and propose a model, named Fábulas—the Portuguese word for *fables*—, that allows authors to create interactive children's eBooks declaratively. Such a higher-level model can then be converted to one of the currently available eBook standard formats or can be directly interpreted by a Fábulas player. The Fábulas model is instantiated as an extension to HTML5 and interpreted in JavaScript, following a polyfill approach, which means it works seamlessly in current modern web browsers.

*DocEng'17, September 4–7, 2017, Valletta, Malta.*
© 2017 ACM. 978-1-4503-4689-4/17/09…$15.00
DOI: http://dx.doi.org/10.1145/3103010.3103016

[1]https://helpx.adobe.com/indesign/using/dynamic-pdf-documents.html

The model discussed in this paper is part of the Fábulas project, which aims at providing an end-to-end platform for writers and educators to create and deliver interactive enhanced children's eBooks, and for children to consume those eBooks. Figure 1 schematically shows the Fábulas platform. The Fábulas platform permeates the three classical hypermedia environments: *authoring*, *storage*, and *exhibition*. Thus, it targets both authors and readers of children's eBooks. Authors interact with the authoring environment using graphical or textual authoring tools to create the eBooks. The eBooks are stored in a document format that conforms to the Fábulas model, and that can be published to the server. The reader uses the Fábulas player to search, read, and interact with the eBooks. This way, the Fábulas document model has to be easily handled by all of those three environments: authoring tools, distribution servers, and players.

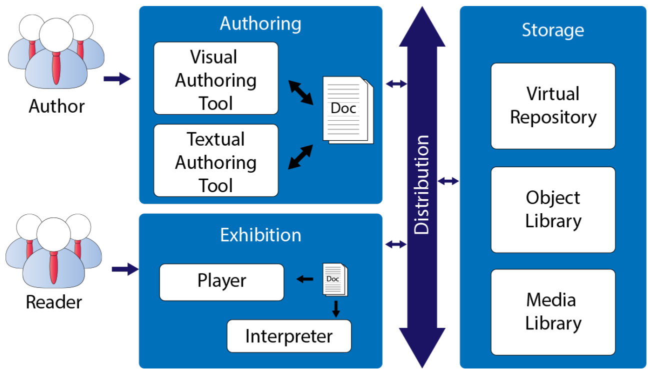

**Figure 1: Overview of the Fábulas platform.**

The remainder of the paper is mainly concerned with the requirements and the model used to create children's eBooks on the Fabulas platform. Section 2 discusses the main requirements for enhanced children's eBooks. Section 3 discusses the related work and compare them to the Fábulas model. Section 4 presents the Fábulas conceptual model. Section 5 details the implementation and usage of the Fábulas model, highlighting its main features through the development of a complete children's story. Finally, Section 6 brings our main conclusions and future work.

## 2 REQUIREMENTS FOR ENHANCED CHILDREN'S EBOOKS

To define the requirements of the Fábulas model, we have conducted a literature review on works trying to define similar requirements. Then, to validate and expand the literature review, we proceeded with a field study in which we analyzed current enhanced children's eBooks available on the Web and in popular application stores. From the literature review and the field study, we extracted the most common features on enhanced children's eBooks, and used them as the basis to the proposed model (detailed in Section 4).

From the literature review, more specifically from [6, 12, 14, 27, 30] we have extracted the following features:

- **Embedded media**, such as audio, video, and image, can be very helpful in engaging children on stories. Audio, for instance, can be used in different ways: additional effects (e.g. synchronized with the presentation, or as a response to a user interaction), background music, narration, and voice recording [12]. In special, **audio narration (or read-aloud) feature** can be used to improve children's vocabulary and can increase comprehension of the story [14];
- **Narration overlays**, i.e., highlighting the text synchronized with the narration, can also be useful, mainly for children that are still learning to read;
- **Animations and in-page interactions**, when applied in the right way, can enrich the children's eBooks and help to capture the attention of children [26]. In special, **reactive animations** may be interesting to present visual cues as a response to user interaction;
- **Alternative story paths (or non-linear narrative)** allows children to feel empowered, and engage them by making their decisions important for the story progression. It also offers greater re-playability and stimulates exploration and curiosity;
- **Auto-play narrative** can be found in some types of children's eBooks and allow the automatic reproduction of the narrative (with or without audio narration);
- **Zoom** is another important feature, which offers children the possibility to take a closer look into details. It can be both **page zoom**, which allows children to see image details or interact with smaller enhanced elements, or **text-only zoom**, which allows increasing/decreasing the text size;
- **Single-page view** is preferred for interactive children's eBooks, whereas in other eBooks it is common to show two pages at a time;
- **Orientation rotation** of eBooks usually present two orientation options, landscape or portrait, and can respond to the position of the device or be fixed in the code.

In the field study, the eBooks were gathered from the Annenberg Learner [2] website and the book apps from Google Play [3] and Apple Store [4]. The criteria we used to select the eBooks in the field study were:

- it should be targeted for children in the age from 6- to 12-years-old;
- it should be an enhanced eBook, which means it should have interactivity, embedded media, or animation features;
- it should have at least 1000 downloads in the respective App stores;
- it should have some feature that was not found in the other selected children's eBooks.

Table 1 summarizes the features we have found in the analyzed eBooks. Those features are presented from the point of view of developers. And, they can be used as requirements to guide the development of authoring models and tools for interactive children's eBooks domain. Thus, this domain knowledge is used in the next section as guidelines governing the creation of the Fábulas conceptual model.

[2]https://www.learner.org/interactives/story/cinderella.html
[3]https://play.google.com/store/apps
[4]https://itunes.apple.com/

**Table 1: Features found on the analyzed enhanced children's eBooks.**

| Features | *Elements of a story (Cinderella)* [5] | *Tales with Gigi (Cinderella)* [6] | *Classic Fairy Tales (Cinderella)* [7] | *Livro mágico/-Magic Book (O galinho Zé)* [8] | *The Jungle Book* [9] | *iStoryBooks (Seeing beyond the obvious)* [10] | *The Little Mermaid* [11] | *Alice for the iPad* [12] |
|---|---|---|---|---|---|---|---|---|
| **Narrative** | | | | | | | | |
| alternative story path | | | | x | | | | |
| multi-language support | | x | x | | x | | x | |
| narration | x | x | x | | x | x | x | |
| narration overlay | | | | | | x | x | |
| auto-play | x | x | x | | | x | x | |
| zoom | | | x | | | x | x | x |
| Single page view | x | x | x | | x | x | | x |
| Orientation rotation* | l | l | l | l | l | p | l | p |
| **Embedded media** | | | | | | | | |
| image | x | x | x | x | x | x | x | x |
| audio | x | x | x | | x | x | x | x |
| text | x | x | x | x | x | x | x | x |
| video | | | | | | | | |
| animation | x | x | | | x | | x | x |
| **Transitions** | | | | | | | | |
| page transitions | | x | x | | x | x | x | x |
| media transitions | x | x | | | x | | x | x |
| **Interaction forms** | | | | | | | | |
| click/touch/gestures | x | x | x | x | x | x | x | x |
| accelerometer | | | | | | | | x |
| ***Widgets*** | | | | | | | | |
| games | x | x | x | | | | x | |
| quizzes | x | | | | | | | |

* Orientation: p = portrait; l = landscape.

## 3 RELATED WORK

Since the 90's, there has been much effort on creating models and standards for interactive multimedia presentations [22]. Some of the most prominent technologies are SMIL [5], NCL [29], MPEG-4 XMT [18, 25], X3D [11], SVG [13], and HTML5 [8]. Nowadays, HTML5 is becoming the ubiquitous solution and is available in almost every device. The research and concepts developed by the other mentioned document models, however, are still useful for defining many types of interactive multimedia applications, and today most of them can be supported in modern web browsers through a polyfill approach. In such an approach, HTML5 can be extended and players for other multimedia languages can be implemented on top of the browser using JavaScript and APIs such as WebGL [20], WebSockets [16], etc. Current examples of those solutions are SMIL TimeSheets [7], Time Style Sheets [19], WebNCL [24], and X3DOM [3]. In this paper, we follow a similar path by defining the Fábulas model and implementing it so that we can reuse the browser infrastructure and extend HTML5 with the concepts of our application domain.

One could argue that for our application domain, authors could directly use HTML5 (plus JavaScript) itself or one of the other document formats discussed above (plus some script language). Indeed, this is true, and most of the features needed by enhanced eBooks can be modeled in lower-level approaches using one of the above document models together with a scripting language. However, those document models are too general and the abstractions they provide are not always closely related to the concepts on interactive children's stories. Different from those general document formats for interactive multimedia presentations, our goal is the design of a minimal and restrictive model closely related to our application domain, i.e., interactive enhanced children's eBooks. (Meixner and Kosch, for instance, follow a similar path for the application domain of hypervideos [23].) After modeling the application using Fábulas, authors can then use a player that natively interpret the Fábulas specification or convert it to one of the lower level document model for multimedia presentations.

eBooks can be seen as a specialized class of interactive multimedia presentations. In its basic form, an eBook is only a sequence of pages, each page containing text and possibly fixed positioned figures. As aforementioned, more advanced eBooks, however, may also provide interactivity and non-linearity, embed other media types such as audio and video, and support animations, quizzes, games, etc. As previously mentioned, examples of eBook formats include PDF, MOBI, and ePub3.

PDF started as a binary format mainly for print fixed-layout documents, including only texts, fonts, and graphics. PDF, however, has evolved to support interactive elements such as annotation, form fields, audio, video, flash animation, and interactive 3D objects. This format is known as *Rich Media* or *Dynamic Media* PDF [17].

ePub is the open source standard for eBooks maintained by the *International Digital Publishing Forum* (IDPF)[13]. Different from PDF, the ePub 3.0 also support eBooks with flow layout, allowing that a page can be adapted to different devices and reading softwares. The current version of ePub, named ePub 3.0, is based on HTML5 and it allows to use multimedia objects (e.g. `<audio>` and `<video>`) features that were not supported in previous versions. The ePub 3.0 specification is divided in four main parts: *EPUB Publications*, *EPUB Content Documents*, *EPUB Open Container Format* e *EPUB Media Overlays*. From those, *Media Overlays* allows synchronizing the eBook text with additional audio. These features are based on the Daisy Talking books [10] standard, which was first created to support accessible books.

Some of the above eBook formats when extended with imperative languages—e.g., ePub3 is extended through JavaScript—can support the requirements of children's eBooks discussed in Section 2. However, we believe that for some recurrent structures on interactive children's eBooks—e.g., animation, in-page interactions, and reactive animations—there should be high-level declarative constructs, which are currently not present in those formats. The Fábulas model try to fill this gap by providing higher-level concepts that are close to the interactive enhanced children's eBook domain. The concepts proposed by the Fábulas model can be integrated in the general eBook formats above, or an interactive story specified in Fábulas can be converted to the lower-level eBook format standards. The next section details the Fábulas model.

## 4 FÁBULAS

Figure 2 presents the Fábulas conceptual model. The root concept of the Fábulas model is the *Fable*. A Fable represents a whole interactive story and is structured as one or more *Chapter*s. Each *Chapter* may contain one or more *Page*s. Also, a *Fable* may contain no chapter, in which case it is composed only of *Page*s, or may be composed of a combination of both *Chapter*s and *Page*s. Each *Page* of the story can be composed of different *media objects* and *agents*. The supported media objects are the conventional media objects found in HTML5, such as image, video, and audio. Also, as will be discussed in Section 5, the supported media types can be extended through a well-defined API. For instance, by default, the Fábulas model already supports an *animation* media object that is composed of a sequence of images that must be presented during a specific duration. An agent allows grouping multiple media object (and other agents) together with their behavior, as a reusable component.

The behavior of the media objects and the nonlinear narrative flow are defined through *Event-Condition-Action* (ECA) rules [21], i.e., each rule defines that when an event is detected, the system evaluates a condition, and if the condition is satisfied, the system execute the action(s). The *events* that can be used to trigger actions are the transitions on media objects or agent states, or user generated events, such as touching an object. The condition part may test if a property (a global, chapter, page, or agent property) is true. Finally, the supported actions may change the state of the presentation (e.g., changing the current page) or the state of media objects or agents.

The remainder of this section details the main concepts above, specifically: *media objects* and *agents*, and the supported *properties*, *events*, and *actions*.

### 4.1 Media objects and Agents

As previously mentioned, each *Page* may be composed of multiple media objects (e.g., text, images, video, etc.). Moreover, an important concept that differs Fábulas from related work is the *Agent* concept. In the Fábulas model, agents are a mechanism to encapsulate the media presentation objects and their associated behavior in the same object. Agents are inspired by the NCM (Nested Context Model) [28] event state machine. However, NCM has the same event state machine to control all medias. In Fábulas model, an agent can define its states and the transitions between them. The agent state machine is defined by a 5-tuple $SM =< S, B, s_0, \Sigma, \delta >$, where:

- $S$ is a finite nonempty set of states
- $B$ is the base composition
- $s_0$ is the initial state
- $\Sigma$ is a set of events
- $\delta : S \times \Sigma \rightarrow S$ is the transition function

Each element of the 5-tuple has its corresponding tags in the Fábulas model. Table 2 shows the equivalence between the elements in the 5-tuple above and the elements in the Fábulas model.

**Table 2: Equivalence between the agent state machine and the elements in the Fábulas model.**

| Agent SM | Fábulas model (example) | Description |
|---|---|---|
| $S$ | <state id="S">...</state> | definition of a state, child of <agent/> |
| $B$ | <img src="..."/> | media objects children of the agent |
| $s_0$ | <state id="$s_0$">...</state> | first state defined in `<agent>` |
| $\Sigma$ | <on-touch>...</on-touch> | events are children of an `<agent>` or `<state>` |
| $\delta : S \times \Sigma \rightarrow S$ | <changeto target="S"/> | child of <on-*/> |

[13]http://idpf.org

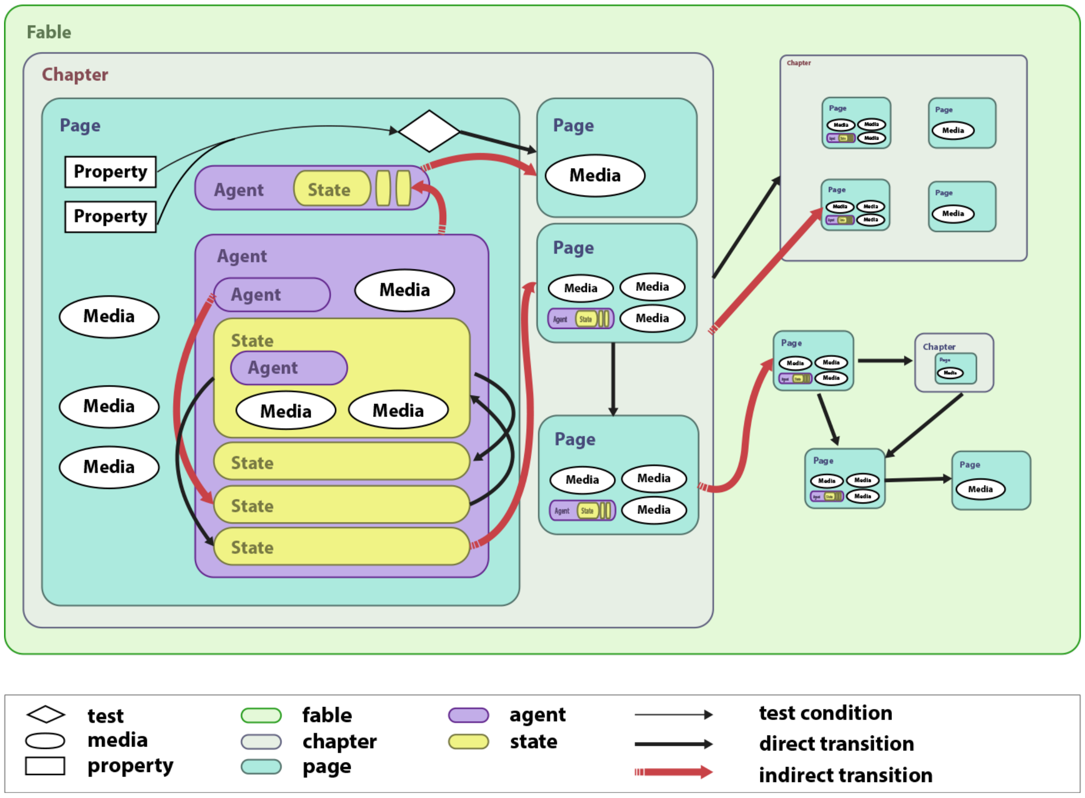

**Figure 2: The Fábulas conceptual model for developing interactive children's eBook.**

An agent state can group media objects and other agents and handle events emitted within its own composition or global events, e.g., user-generated events. The basic state of an agent is composed of the media objects defined as its direct descendent. At execution time, those elements are part of all the other presentation states. By default, the initial state of an agent is the first one defined by an `<agent>` element. The state of an agent can be transitioned by the *changeto* action, informing the next state.

Figure 3 shows an agent representing a "locked door" containing an internal "switch" agent, which, thanks to its self-contained definition, can be reused in different situations. Listing 1 shows a possible implementation of the agents in Figure 3. The "switch" agent (lines 2–13) and the event observation `<on-capture>` (lines 31–33) compose the basic state of the agent "door-with-lock". The "switch" agent contains two states "locked" and "unlocked". It starts in the "locked" state and, when clicked by the user, it goes to the "unlocked" state, and emits a custom event named "unlock" (lines 5–8). This custom event is emitted in the scope of the page in which the agent is inserted. The initial state of the "doorwithlock" agent is named "locked-door" (lines 15–17) and is composed of one image only. When the event "unlock" is emitted this agent goes from the "locked-door" to the "unlocked" state (lines 18–23). When the user touches the agent in "locked-door" state, it goes to the "open" state (lines 24–29). And, if the user touches the agent in "open" state, the agent emits a *end_page* event (line 27) asking the system to go to the next page.

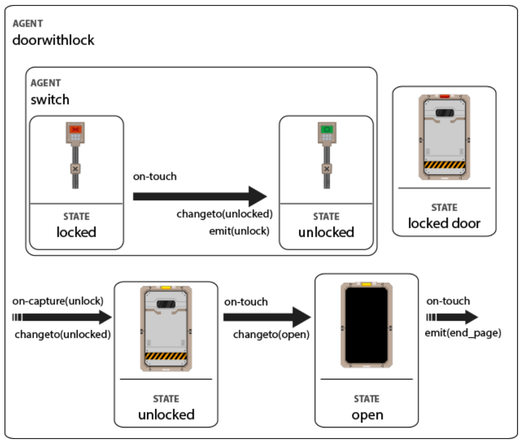

**Figure 3: Agent example of a locked door.**

[7]https://www.learner.org/interactives/story/cinderella.html
[8]https://play.google.com/store/apps/dev?id=6290881108498971005
[9]https://play.google.com/store/apps/details?id=es.cuentuvere.cuentosclasicos
[10]https://play.google.com/store/apps/details?id=br.com.fabricadesonhos.applivromagico
[11]https://play.google.com/store/apps/details?id=air.com.goodbeans.JungleBookStoryBook
[12]https://play.google.com/store/apps/details?id=com.infomarvel.istorybooks
[13]https://play.google.com/store/apps/details?id=com.storytoys.TheLittleMermaid.GooglePlay

```
<agent id="doorwithlock">
  <agent id="switch">
    <state id="locked">
      <img src="switch-locked.png"/>
      <on-touch>
        <changeto target="unlocked"/>
        <emit event="unlock" scope="page"/>
      </on-touch>
    </state>
    <state id="unlocked">
      <img src="switch-unlocked.png"/>
    </state>
  </agent>

  <state id="locked-door">
    <img src="locked-door.png"/>
  </state>
  <state id="unlocked">
    <img src="locked-door.png"/>
    <on-touch>
      <changeto target="open"/>
    </on-touch>
  </state>
  <state id="open">
    <img src="open-door.png"/>
    <on-touch>
      <emit event="end_page"/>
    </on-touch>
  </state>

  <on-capture event="unlock">
    <changeto target="unlocked"/>
  </on-capture>
</agent>
```

**Listing 1: Example of an agent specified using Fábulas.**

## 4.2 Properties

Pages, media objects, and agents may have a set of associated properties. The supported properties are sensitive to the object type. For instance, audio media objects have a volume property, which is not present in purely graphical media object, such as images. The properties are grouped into *system*, *object*, *physics*, and *user-defined* properties. System properties represent global features of the system, such as *lightLevel* and *soundLevel*. System properties can be inherited based on the scope of a composition, e.g., the *soundLevel* set for one page does not interfere with the volume set for other pages. Object properties directly define the presentation characteristics of the media objects or agents, e.g., width, height, transparency, etc.. Physics properties assign characteristics that can define the behavior of media objects when interacting with the enviroment; through pyshics properties, it is possible to simulate weigth, collision, and mobility for objects (see Table 3). Finally, the user-defined properties are custom-made properties defined by the author, and can be used as the author needs.

**Table 3: List of the physics properties currently supported on Fábulas.**

| type | Name | Description |
|---|---|---|
| *physics* | solid | two *solid* media objects generate collision events when their borders touch |
| | heavy | add heavy mobility to the media object |
| | light | add light mobility to the media object |
| | draggable | user can drag the media object |

## 4.3 Events

The default events supported by Fábulas (summarized on Table 4) can be divided into: *presentation* events, *interaction* events, and *narrative control* events.

Presentation events are those that reflect a change in the state of the pages, media objects, or agents on the narrative, e.g., the beginning of an audio presentation.

Interaction events are events that involves some user activity or the interaction between two or more objects of the narrative, e.g., click or gestures.

Narrative control events are those used by the user to control the narrative flow, e.g., go to the next page; in special, they can be used to define nonlinear narratives.

**Table 4: Fábulas model events.**

| Type | Name | Description |
|---|---|---|
| presentation | begin | the begining of a media object |
| | end | the end or natural end of a media object |
| | pause | pause the presentation of a media object |
| | continue | resume the presentation of the media |
| | show | a media object is shown |
| | hide | a media object is hidden |
| interaction | touch* | user touches the media object or agent |
| | drag | user drags a media object or agent |
| | drop | user drops the media object or agent |
| | swipe** | user performs a swipe gesture on the screen |
| | tilt** | user performs a tilt on the device |
| | pinch** | user performs a pinch gesture on the screen |
| | collision*** | collision between media objects |
| control | capture | capture custom events |
| | end_page | end of the presentation of the current page |
| | end_chapter | end of the presentation of the current chapter |
| | the_end | end of the history |
| | previous_page | user moved to previous page |
| | previous_chapter | user moved to the previous chapter |

* equivalent to the mouse click
** depends on the device
*** needs the property solid

By default, a Fábulas player presents the pages in the order they appear in the story definition, which eases the development of linear narratives. By capturing narrative control events, it is possible to define additional navigation rules through the ECA paradigm. The default ordering of the pages, and the navigation rules allow authors to create alternative paths on the presentation flow, and to create more complex narratives. For instance, the `previous_page` event prescribes that the current page must be ended and the one that was being previously rendered must be presented again. The `previous_page` events works as a stack that saves the ordering in which the pages were visited, according to the followed narrative flow. In the first page the `previous_page` event has no effect.

### 4.4 Actions

Actions change the state of the objects in the narrative. Table 5 shows the actions supported by the Fábulas model.

**Table 5: The actions supported by the Fábulas model.**

| Type | Name | Description |
|---|---|---|
| presentation | start | starts the presentation of a media object |
| | stop | finishes the presentation of a media object |
| | pause | pauses a media objects |
| | resume | resumes the presentation of a previously paused media object |
| | show | shows a media object |
| | hide | hides a media object |
| control | emit | emits a custom user event |
| | changeto | changes the *state* of an agent, a chapter, or a page |
| | set | changes the value of a property |

The action `emit` fires an internal event to the execution machine. This event may be used as a trigger to other actions in other parts of the fable. The author can inform the scope for the propagation of the event, as one of the values: *fable*, *chapter*, or *page*. The *fable*, *chapter*, and *page* scopes, respectively, refers to all the elements in the fable, in the current chapter, or in the current page will be notified of the event. The agents, even those that are not currently active (such as agents in other pages, not presented yet) can be notified of events.

`Start` and `stop` actions can be used for all types of media. `Pause` and `resume` are actions usually applied to continuous media types, such as, video, audio, and animations. The `show` and `hide` actions are aimed mainly for graphic media object. The `hide` action, for instance, can hide temporal media objects such as videos, but it does not interrupt its execution. In the case of other media types, such as images, it presents the same result of the `start` and `stop` actions.

## 5 INTEGRATION OF FÁBULAS ON THE WEB

As aforementioned, the Fábulas model is currently integrated into the Web using a polyfill approach. The polyfill approach is especially useful for rapid prototyping because it relies on the multimedia execution machine of the browser, only extending what is needed, with the help of JavaScript libraries. In our implementation we use Angular.js [1]. Angular.js has an HTML compiler that allows the developer to define new syntax for default HTML5 elements or create new elements. Table 6 shows the elements we have created (or modified) for the implementation of the Fábulas model on HTML5.

**Table 6: The elements of the Fábulas model.**

| Element | parent | attributes* |
|---|---|---|
| fable | - | id, src, width, height, bg-img, bg-sound, bg-sound-rep |
| chapter | fable | id, src, bg-img, bg-sound, bg-sound-rep, trans-in, trans-out |
| page | fable, chapter | id, src, bg-img, bg-sound, bg-sound-rep, trans-in, trans-out |
| property | fable, chapter, page | name, value |
| img** | page, div, agent, state | left, top, right, bottom, width, height |
| div** | page, div, agent, state | left, top, right, bottom, width, height |
| p** | page, div, agent, state | left, top, right, bottom, width, height |
| span** | page, p, div, agent, state | - |
| audio** | page, agent, state | - |
| video** | page, agent, state | left, top, right, bottom, width, height |
| animation | page, agent, state | id, left, top, right, bottom, width, height |
| agent | page, agent, state | id, left, top, right, bottom, width, height, draggable, heavy, light, solid |
| state | agent | id |
| on-(event)*** | fable, chapter, page, agent | event****, target, test, delay |
| (action)*** | on-(event) | event****, target, delay, value |

* all elements contains the basics HTML attributes
** based on the HTML elements
*** event/action name
**** in case the capture event or emit action

In the Angular.js nomenclature, a *directive* is defined as a behavior that must be triggered when specific HTML constructs are found during the building process. These directives can be elements, attributes, class names, or comments.

As an example of defining a directive in Angular.js, let us take the source code on Listing 2 and Listing 3.

Listing 2 shows a code snippet containing a `<div>` element with the attributes *left*, *top*, *width*, and *height*, which are redefined by our implementation. Those attributes together with the *right* and *bottom* attributes (also redefined by our implementation) specify the position of a media object or agent.

Listing 3 shows the JavaScript code with the *left* attribute directive. (The other attributes directives are similar, so they are omitted here.) By defining that directive, when the page is loading, all the elements that have a *left* attribute will have the behavior described by the function of the *link* parameter (line 4). The type of the directive (element, attribute, class name, or comment) is constrained by the parameter *restrict* (line 3). In the example, the code affects any element that contain a *left* attribute. More complex combination can also be created using JavaScript functions, such as, “all the elements with tagname *x* that have an attribute *y*”, and so on.

```
<div class="text" id="title" width="250" height="50" top="50"
      left="50">
  The Little Knight
</div>
```

**Listing 2: Example showing the usage of the attributes *left*, *top*, *width*, *height*.**

```
fablePlayer.directive('left', ['$document', function(
      $document) {
   return {
     restrict: 'A',
     link: function (scope, element, attr) {
       element.css({
         left: attr.left + 'px'
       });
     }
   };
}]);
```

**Listing 3: Definition of the Angular.js directive for the attribute *left*.**

Our implementation takes advantage of the above-discussed mechanism provided by Angular.js to define all the elements and attributes of Table 6. This way, we can seamlessly integrate the Fábulas concepts into HTML5 webpages. From the authors viewpoint, they can use the Fábulas elements directly into HTML5 documents. To exemplify this integration, the next subsection discusses a concrete interactive story using web integrated Fábulas model.

## 5.1 Usage example

The interactive story discussed in this section, named “The Little Knight”, is about a knight who, upon receiving the news of a friend with whom he had long lost contact, sets out on a journey to try to help him. Figure 4 shows the story flow of “The Little Knight”, and the linear order in which the pages are defined in the story. The story unfolds in five pages. Four pages are part of the main story, and the last one is an alternative path. The main features contained in the story are images, audio, user interaction, navigation, and conditional tests.

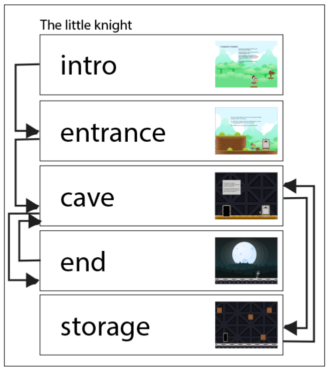

**Figure 4: The story flow of “The Little Knight”.**

Listing 4 shows the source code of the required header and of the first page of the story, named “intro”.

Since we use Angular.js, first, we need to import the library, the player code, and the default style for the application (lines 6–8). It is also needed to add the *ng-app* and *ng-controller* attributes on the page to elements that will contain the `<fable>` element. In the example, they were placed in the `<html>` and `<body>`, respectively.

On the first page (lines 14–32) it is possible to note some of the Fábulas elements and attributes. For instance, the *bg-img* and *bg-sound* attributes on the `<page>` elements represent the background image and the ambient music for the page, respectively. For simple media types—such as text, image, audio, and video—the standard HTML tags are used. Moreover, all the elements can be stylized by CSS stylesheets, as in standard HTML5 pages. Besides simple media objects, this first page also defines an agent, the “board” agent (lines 17–24), which represents a board that, when touched by the user (`<on-touch>`, fires an event (`<emit>`) that ends the presentation of the current page and advances to the next ones (lines 20–22).

```
1  <!DOCTYPE html>
2  <html lang="en" ng-app="fablePlayer">
3    <head>
4      <meta charset="UTF-8">
5      <title>The little knight</title>
6      <script src="angular.min.js"></script>
7      <script src="fablePlayer.js"></script>
8      <link rel="stylesheet" href="fable-player.css">
9      <link rel="stylesheet" href="little_knight.css">
10   </head>
11   <body ng-controller="fablePlayerController">
12     <fable width="800" height="600">
13       <property name="hasantidote" value="false"></property>
14       <page id="intro" bg-img="BG.png" bg-sound="audio1.mp3">
15         <div class="text" id="title" width="250" height="50"
                top="50" left="50">O pequeno Cavaleiro</div>
16         <img src="char1.png" height="180" width="110"
                bottom="180" left="440"/>
17         <agent id="board" top="370" right="140" width="105"
                height="110">
18           <state>
19             <img src="board.png"/>
20             <on-touch>
21               <emit event="end_page"/>
22             </on-touch>
23           </state>
24         </agent>
25
26         <div width="200" height="300" top="100" left="300">
27           <p>Once upon a time there was a small knight who
                received a letter from a friend.</p>
28           <p>He was very very very surprised, because this
                friend was gone for years.</p>
29           <p>Except the news on the letter was not very good.
                 His friend had contracted a mysterious illness
                and needed help to get the cure.</p>
30           <p>As fast as he could, he went to meet this
                friend.</p>
31         </div>
32       </page>
33       ...
34     </fable>
35     ...
36   </body>
37 </html>
```

**Listing 4: Example of a story in an HTML page.**

The second page, “entrance”, is mainly composed of agents: one “board”, one “door”, one “bush with the key”, three “bushes without a key” agents.

Listing 5 shows the part of code of the “entrance” page that contains the “board” agent. When this agent is touched by the user, it shows a warning with a tip about where he will find the key to open the door. Lines 3–6 show a simple animation of the board while it is in state (`<state>`) “swinging”. When the board is in the “swinging” state and the user touches it (`<on-touch>`), it transitions to the “warning” state. After 3 seconds—due to the *delay* in the *changeto* action (line 9)—or when it is touched again (lines 17–18) it returns to “swinging” state.

Listing 6 shows the source code of the “bush with the key” agent and Figure 5 schematically shows its behavior, and how it interacts with the “door” agent. The main difference between the “bush with the key” agent, and the three others “bushes without a key” is the internal “key” agent (lines 11–19). When the “key” agent is touched, it emits the event “gotKey”. The “gotKey” event is captured by the “door” agent.

```
 1  ...
 2  <page id="entrance">
 3    <agent id="placa" top="510" right="230" height="50"
          width="50">
 4      <state id="rebolando">
 5        <animation dur="0.5" rep="indefined">
 6          <img src="board1.png" height="50" width="50"/>
 7          <img src="board2.png" height="50" width="50"/>
 8        </animation>
 9          <on-touch>
10          <changeto target="warning"/>
11          <changeto delay="3" target="animation"/>
12        </on-touch>
13      </state>
14      <state id="aviso">
15        <img src="board1.png" height="50" width="50"/>
16        <div class="warning">
17          The key is on the bushs.
18        </div>
19        <on-touch>
20          <changeto target="animation"/>
21        </on-touch>
22      </state>
23    </agent>
24    ...
25  </page>
26  ...
```

**Listing 5: Code of the “board” agent, in the “entrance” page.**

```
 1  <page id="entrance">
 2    ...
 3    <agent id="bush" top="310" left="50">
 4      <state id="closed">
 5        <img src="bush.png" height="46" width="73"/>
 6        <on-touch>
 7          <changeto target="open"/>
 8        </on-touch>
 9      </state>
10      <state id="open">
11        <img src="openbush.png" height="50" width="50"/>
12        <agent id="key" top="0" left="10">
13          <state>
14            <img src="key.png" height="30" width="30"/>
15            <on-touch>
16              <emit event="gotkey" scope="page"/>
17              <stop target="key"/>
18            </on-touch>
19          </state>
20        </agent>
21      </state>
22    </agent>
23    ...
24  </page>
```

**Listing 6: Source code of the bush that contains the key, in the “entrance” page.**

Finally, still on the “entrance” page, Listing 7 shows the source code of the “door” agent. Lines 6–8 defines what happens when the event *on-capture* happens: the agent emits the “gotKey” event, and then transitions to the “unlocked” state. When the agent is in the “unlocked” state and it is touched (`<on-touch>`) the door emits the event end_page, that informs that the presentation of the current page must be finished and the next page must be shown.

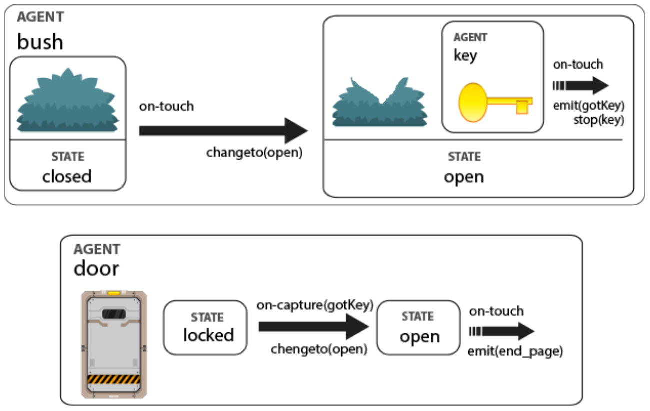

**Figure 5: Agent “bush” and agent “key” interacting with agent “door”.**

```
 1  ...
 2  <page id="entrance">
 3    ...
 4    <agent id="door" bottom="50" right="100" height="150"
          width="100">
 5      <img src="DoorLocked.png" height="150" width="100"/>
 6      <state id="locked">
 7        <on-capture event="gotkey">
 8          <changeto target="open"/>
 9        </on-capture>
10      </state>
11      <state id="open">
12        <on-touch test="hasKey">
13          <emit event="end_page"/>
14        </on-touch>
15      </state>
16    </agent>
17    ..
18  </page>
19  ...
```

**Listing 7: Source code of the “door” agent, in the “entrance” page.**

In the third page, named “cave”, the reader can choose between two different paths on the story: the first one goes to the “storage” page; and, the second goes to the “end” page. Listing 8 shows the two agents, representing the two doors, that the reader can use them to choose his path. The “storage-path” event (line 7) is captured by the page (lines 37–39) that then changes to the “storage” page. The “endDoor” agent (lines 11–35) issues the event end_page (line 29) that closes the current page and goes to the next one in the sequence, i.e., the “end” page.

In the “storage” page—which source code is not shown here mainly for brevity, since, in principle, it uses similar contructs already discussed—the reader can find the antidote for the friend of the knight in one of the three “boxes” agents. Those agents works very similarly to the “bushes with/without the/a key”. When the user finds this antidote he changes the “hasAntidote” variable to *true*. In the “end” page, the “hasAntidote” variable is tested, and if its value is *true*, then, the friend of the knight will be cured, and the story ends; otherwise, the reader has to go back to look for the antidote on the “storage” page.

```
<page id="cave" bg-img="bg3.png">
  ...
  <agent id="storageDoor" bottom="30" left="100" height="150"
        width="100">
    <state>
      <img src="DoorOpen.png" height="150" width="100"/>
      <on-touch>
        <emit event="storage-path" scope="page"/>
      </on-touch>
    </state>
  </agent>
  <agent id="endDoor" bottom="30" right="100" height="150"
        width="100">
    <img src="DoorLocked.png" height="150" width="100"/>
    <agent id="switch">
      <img src="button.png" left="-30" width="20" height="70"
            bottom="0"/>
      <state id="locked">
        <on-touch>
          <emit event="unlock" scope="page"/>
          <changeto target="unlocked"/>
        </on-touch>
      </state>
      <state id="unlocked">
      </state>
    </agent>
    <state id="locked">
    </state>
    <state id="open">
      <img src="DoorOpen.png" height="150" width="100"/>
      <on-touch>
        <emit event="end_page"/>
      </on-touch>
    </state>
    <on-capture event="unlock">
      <changeto target="open"/>
    </on-capture>
  </agent>

  <on-capture event="storage-path">
    <changeto target="storage-path"/>
  </on-capture>
</page>
```

**Listing 8: Source code of the "cave" page.**

## 6 CONCLUSION

In this work, we discussed the main requirements of interactive eBooks for children and we presented the Fábulas conceptual model for allowing authors to create them. The proposed conceptual model is instantiated in XML as an extension to HTML5 and following a polyfill approach, which allows a seamless integration with modern web browsers. Besides presenting the elements of the XML instantiation of Fábulas model, we also show a complete example that uses the main features of the model—in special, animation, object states control, and nonlinear narrative definition.

Although our current implementation, discussed in Section 5, is complete enough for supporting many types of enhanced interactive eBooks for children, it still does not fully support all the requirements discussed in Section 2 through high-level abstractions. Even though those requirements can be currently achieved by a lower-level approach using JavaScript, we also plan to provide abstractions for them in the future. In special, the following requirements will be the focus of extensions on the model: narration and narration overlay (e.g., using the media overlay from ePub3); multi-language support; complex animations; and, widgets.

Other future works include: the definition and formalization of guidelines and a complete process for authoring enhanced children's eBooks; and the integration of the concepts of the Fábulas model (e.g., the concept of agents) into the ePub3 document format. Finally, as previously mentioned, the model discussed in this paper is part of a bigger project that includes all the steps in the creation, distribution, and consumption of interactive eBooks for children. Thus, another future work, already initiated, is the development of an authoring tool based on the Fábulas model, which will allow authors to create and upload eBooks with a graphical authoring tool, easily reusing components developed by other authors.

## REFERENCES

[1] Angularjs: Developer guide. https://docs.angularjs.org/guide. Accessed: 2017-04-10.

[2] Mobi. https://wiki.mobileread.com/wiki/MOBI. Accessed: 2017-04-10.

[3] J. Behr, P. Eschler, Y. Jung, and M. Zöllner. X3dom: A dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[4] S. M. Benedetti. Ebook interativo: hipermídia no livro eletrônico. 2016.

[5] D. C. Bulterman and L. W. Rutledge. *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer Publishing Company, Incorporated, 2008.

[6] A. G. Bus, Z. K. Takacs, and C. A. Kegel. Affordances and limitations of electronic storybooks for young children's emergent literacy. *Developmental Review*, 35:79 – 97, 2015. Special Issue: Living in the "Net" Generation: Multitasking, Learning, and Development.

[7] F. Cazenave, V. Quint, and C. Roisin. Timesheets.js: When smil meets html5 and css3. In *Proceedings of the 11th ACM Symposium on Document Engineering*, DocEng '11, pages 43–52, New York, NY, USA, 2011. ACM.

[8] J. Choi, Y. Lee, and K. Kim. Html5 based interactive e-book reader. *International Journal of Software Engineering and Its Applications*, 8(2):67–74, 2014.

[9] G. Conboy, M. Garrish, M. Gylling, W. McCoy, M. Makoto, and D. Weck. EPUB 3.1 overview. Informational document, International Digital Publishing Forum (IDPF), Jan. 2017. http://www.idpf.org/epub3/latest/overview.

[10] D. Consortium et al. Daisy/niso standard. ansi/niso z39.86 specifications for the digital talking book, April 2012.

[11] L. Daly and D. Brutzman. X3d: Extensible 3d graphics standard [standards in a nutshell]. *IEEE Signal Processing Magazine*, 24(6):130–135, Nov 2007.

[12] K. M. Doty. Designing for interactive ebooks: an evaluation of effective interaction elements in children's ebooks, 2015.

[13] J. Ferraiolo, F. Jun, and D. Jackson. Scalable vector graphics (svg) 1.1 specification, w3c recommendation 14 january 2003. *URL: http://www. w3. org/TR/SVGll*, 2003.

[14] S. Grimshaw, N. Dungworth, C. McKnight, and A. Morris. Electronic books: Children's reading and comprehension. *British Journal of Educational Technology*, 38(4):583–599, 2007.

[15] M. Hart. *Project gutenberg*. Project Gutenberg, 1971.

[16] I. Hickson. The websocket api. *W3C Working Draft WD-websockets-20110929, September*, 2011.

[17] A. S. Incorporated. Pdf reference, sixth edition: Adobe portable document format version 1.7., November 2006.

[18] M. Kim, S. Wood, and L.-T. Cheok. Extensible mpeg-4 textual format (xmt). In *Proceedings of the 2000 ACM Workshops on Multimedia*, MULTIMEDIA '00, pages 71–74, New York, NY, USA, 2000. ACM.

[19] R. Laiola Guimarães, D. Bulterman, P. Cesar, and J. Jansen. Synchronizing web documents with style. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, WebMedia '14, pages 151–158, New York, NY, USA, 2014. ACM.

[20] C. Marrin. Webgl specification. *Khronos WebGL Working Group*, 2011.

[21] D. McCarthy and U. Dayal. The architecture of an active database management system. *SIGMOD Rec.*, 18(2):215–224, June 1989.

[22] B. Meixner. Hypervideos and interactive multimedia presentations. *ACM Comput. Surv.*, 50(1):9:1–9:34, Mar. 2017.

[23] B. Meixner and H. Kosch. Interactive non-linear video: Definition and xml structure. In *Proceedings of the 2012 ACM Symposium on Document Engineering*, DocEng '12, pages 49–58, New York, NY, USA, 2012. ACM.

[24] E. L. Melo, C. C. Viel, C. A. C. Teixeira, A. C. Rondon, D. d. P. Silva, D. G. Rodrigues, and E. C. Silva. Webncl: A web-based presentation machine for multimedia documents. In *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*, WebMedia '12, pages 403–410, New York, NY, USA, 2012. ACM.

[25] F. C. Pereira and T. Ebrahimi. *The MPEG-4 book*. Prentice Hall Professional, 2002.

[26] E. Rhodes and G. Walsh. Recommendations for developing technologies that encourage reading practices among children in families with low-literate adults. pages 125–136, 2016.

[27] B. Sargeant. What is an ebook? what is a book app? and why should we care? an analysis of contemporary digital picture books. *Children's Literature in Education*, 46(4):454–466, 2015.

[28] L. F. G. Soares and R. F. Rodrigues. Nested context model 3.0: Part 1–ncm core. *Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio*, (18/05), 2005.

[29] L. F. G. Soares and R. F. Rodrigues. Nested context language 3.0 part 8–ncl digital tv profiles. *Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio*, 1200(35):06, 2006.

[30] J. Yokota and W. H. Teale. Picture books and the digital world. *The Reading Teacher*, 67(8):577–585, 2014.