Comics2D: Describing and Creating Comics from Story-Based Applications with Autonomous Characters

Tiago Alves[1], Adrian McMichael[2], Ana Simões[1],
Marco Vala[1], Ana Paiva[1], and Ruth Aylett[2]

[1] IST - Technical University of Lisbon, Avenida Professor Cavaco Silva, Tagus Park
2780-990 Porto Salvo, Portugal

[2] MACS, Heriot-Watt University
Riccarton, Edinburgh EH10 4ET

{tiago.alves, ana.simoes}@tagus.ist.utl.pt, {marco.vala, ana.paiva}@inesc-id.pt,

awm1@hw.ac.uk, ruth@macs.hw.ac.uk

ABSTRACT
Comics are a well-known visual medium that is perfectly suited to tell stories. Traditional artists are capable of producing eye-catching and non-repetitive drawings which result in stories told in a very effective way. Our goal is to use comics to display summaries of story-based applications as a way to capture all the deepness of its characters and plots. We identify common concepts in comics, introduce the Comic Strip Description Language (CSDL) and propose a system to automatically generate comic strips using this language. We believe that the expressive power added by the graphics is necessary to create richer summaries of story-based applications and to make the reader experience the personalities and the emotions felt by the autonomous characters in these applications.

Categories and Subject Descriptors
I.3.3 [**Computer Graphics**]: Picture/Image Generation - display algorithms; D.2.11 [**Software Engineering**]: Software Architectures – languages.

General Terms
Algorithms, Design, Languages.

Keywords
comics, webcomics, graphical histories, non-photorealistic rendering, automatic generation.

1. INTRODUCTION
Comics are a form of visual art. Like an album of pictures that depicts all the moments in a trip, comic strips capture all the essence in a narrative and can tell us a story in a very effective way. These properties make comics very attractive as a medium to depict the main happenings in a story-based application. Comics have a deep complexity in each element and also a great expressive power which can convey small but important details. A single comic panel can be as simple as a child's drawing or as sophisticated as a painter's work of art.

Further, when we look at state-of-art narrative systems like Façade [11], Marc Cavazza et al.'s system [3], FearNot! [1] or recent computer games with interesting narrative display like Indigo Prophecy [9], we witness the use of autonomous characters with deeper personalities, complex behaviour and the ability to show inner feelings. Traditional text-based summarization methods [2][13] cannot capture all the subtleness in these stories and comics might fill the gap adding the expressivity of the visual channel to the summary.

Our goal is to automatically generate comics from story-based applications, like the previous examples, in order to create visual story summaries instead of traditional application logs. However, generating comics from stories is a complex process. We need to gather all the necessary information, choose the most relevant events and create individual panels that form the comics strip. When this process is automatic it usually leads to repetitive and unappealing drawings which don't have great impact on the reader.

Another challenge is to make such system independent from the source applications. We need a common language that describes a comic strip in a generic way so that we can generate the summaries using this language and then create the graphical content.

In the next section we look at existent comics description languages and applications for automatic generation of comics. Section 3 depicts the most important comics elements and introduces the

Comic Strip Description Language to describe them. In Section 4 we present an overview of Comics2D, our system to generate comics. Finally, we draw some conclusions and outline future work.

2. RELATED WORK

There are many reasons why a description language for comics might be useful. Be it for cataloguing purposes or to find comics faster and easier on the Internet. Still, describing the full contents of a comic book or a comic strip is no easy task and there are still many hurdles to overcome.

The same applies to creating comics automatically using computers. Although contemporary artists use computers to ease the process of comics production, having computers themselves creating graphical content for comics is a different issue.

2.1 Comics description languages

There are at least two languages to describe comics, namely, the Comic Book Markup Language (CBML) [5] and the Comics Markup Language (ComicsML) [6].

ComicsML is a very simple description language that is intended to work much like an RSS feed for comic strips. Each panel of the comic strip can be described by providing basic information. This information is a link to the already drawn image and a description of the actions that take place. Likewise, the text that the panel contains, be it from narrative or from speech balloons, is also denoted.

The CBML is a more advanced description language. Its aim is to provide a way to digitize comic books so that they can be preserved for a longer time. With this goal in mind, CBML allows to store advanced metadata associated with the respective comic book. Examples include the publisher and the character list. We can also specify information about the characters that are present in each panel and like in ComicsML the text associated with both narrative text and speech balloons.

These two languages are meant to describe comics that already exist. ComicsML focuses on comics from the digital medium and CBML on comics from the traditional medium. However, both languages have a limited notation that fails to describe complex and artistic layouts like overlapping panels or random panel positions.

2.2 Automatic comics generation

One way of automatically creating graphical content for comics is to take pre-drawn images, like characters and backgrounds, and join them to get a final image. We call this method automatic composition.

On the other hand, if we have access to the visual representation of what we want to display in the comic strip, we can simply take a screenshot and enrich its content. This method is called automatic transformation of dynamic graphics.

2.2.1 Automatic composition

Comic Chat [10] is probably the best example of how to render a comic panel by taking individual images and laying them out together. The goal of this system is to depict online conversations occurring in a virtual chat room.

Every chat participant has a comic character that symbolizes him. And every comic character has a finite number of facial expressions and body poses that are independent. To determine the correct expression and pose for each situation, the system does a semantic analysis of the participant's messages. If no emotion is found during this analysis, the system selects a neutral expression and pose.

Comic Chat also has rules for the inclusion and the positioning and orientation of the characters. Although it is recommended not to show every character in every panel, it is important to always show speaking characters. And those who are being spoken to should be at least in the first panel of a multi-panel dialogue. The position and orientation of the characters have to be in accordance to the occurring dialogue. This means that people who are talking to each other should face one another and should not have anyone between them.

In addition to the characters, the system has other pre-drawn elements to add richness and variety to the resulting panel, namely, backgrounds and other visual elements. These are chosen according to the semantic meaning of a phrase or the use of some keyword.

On the contrary, speech balloons are drawn dynamically. The algorithm is fairly simple but good enough to hide the fact that the balloons are computer-generated.

Figure 1. Panel created by Comic Chat [10].

2.2.2 Automatic transformation of dynamic graphics

Ariel Shamir et al. [15] presented a system that is able to represent dynamic graphics in comics art form. The goal of their system is to create a visual summary of a three-dimensional computer game in non-photorealistic rendering quality simulating the look and feel of comics.

The system consists of various parts, the first ones being the *logger*, the *scener* and the *director*. These three subsystems together are responsible for the creation of the story that will be subsequently depicted. First there is the need to log the interactions happening in the game engine. These interactions are then transformed into a coherent story separated into scenes. Finally, the

main interactions are selected by analyzing a function that returns the level of importance of each one of them.

At this stage it is still necessary to convert the 3D scenes into 2D images. This is achieved by the *renderer*, who is primarily responsible for setting up the camera. To accomplish this task, the *director* feeds the *renderer* with high level directives, such as the preferred type of shot, which include direction and zoom-factor. These directives are transformed by the *renderer* to explicit camera parameters. To choose the best shot, many shots are taken around the desired point. The one with the best visibility of the primary entity followed by the best visibility of the secondary entities is chosen.

Further, the *renderer* has to manipulate the newly created images so that they resemble hand-drawn cartoons. This is done in post processing by stylizing the images. Various image processing algorithms are applied to achieve the final result.

The creation of speech balloons is also the responsibility of the *renderer*. He creates simple balloons and splits them along consecutive panels if the text is too big to fit inside a single one. The balloon placement is rather simple, since the game characters seem to appear always at the centre of the screen.

Figure 2. Panels created by Ariel Shamir et al.'s system [15].

Using this approach, the *renderer* needs to implement a camera planning algorithm to be able to take the required shots. Thus, camera planning assumes a vital importance in this type of comics generation.

3. COMICS LANGUAGE

When we look at comics, we usually are not aware of all the elements that are needed to create a single panel. Each of these elements, that most of us take for granted, has its own purpose of either improving the understanding or enriching the comic visually. The books of Will Eisner [7] and Scott McCloud [12] are an invaluable help identifying the concepts behind comics.

After having understood the structure of comics and what they are made of, we were able to define the input specification of our system. As mentioned before, there are other description languages to define comics but they didn't suit the specific need of our system.

3.1 Comic Strip Description Language

The Comic Strip Description Language (CSDL) is an XML-based markup language that semantically describes the story that our system should visually depict. The organization of the document is set up like a comic strip.

The root XML node is the `comic` node where information, such as the comic title, can be defined.

Its children are all `scene` nodes. Their function is to separate the story into different scenes according to temporal and/or spatial distances or just change of subject in the story.

The children of the `scene` nodes are of the type `panel` or `transition`. The `panel` nodes describe what is caught inside a comics panel: an important moment or action of the story. The `transition` node is positioned between two consecutive `panel` nodes to define the panel-to-panel transition that happens at this specific position. It is basically an optional helper node to give additional information since the panel-to-panel transitions are an abstract concept that is implicitly contained in the already defined panels.

According to McCloud [12] there are six different transition types between panels:

- **Moment-to-moment.** This type of transition is used to break a single event into several parts, showing that little time has passed between them. This is useful to build up tension. They are hardly ever used.
- **Action-to-action.** These transitions focus on a single subject that is performing distinct actions. They are by far the most often used transitions.

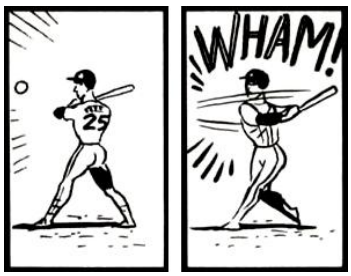

Figure 3. Action-to-action transition. In McCloud [12].

- **Subject-to-subject.** Here, the reader is transported from one subject to another while staying within the same scene. His imagination is very important and it's up to him to give some meaning to the transition.

Figure 4. Subject-to-subject transition. In McCloud [12].

- **Scene-to-scene.** Deductive reasoning is what's required in this transition. They represent a change of scene, transporting the reader across significant distances of time and space.

Figure 5. Scene-to-scene transition. In McCloud [12].

- **Aspect-to-aspect.** When an artist wants to show different aspects of a place or idea while avoiding to show how time is passing, he chooses this type of transition. However, it is rarely used.

Figure 6. Aspect-to-aspect transition. In McCloud [12].

- **Non-sequitur.** This transition type is also seldom used. It offers no logical relationship between panels.

The `panel` node has three attributes. The first one is the border type, the second is the duration of the happenings depicted in the respective panel, and the third defines the panel's importance. The two last attributes help the system to choose a technique to show how much time the panel portrays and if it should stand out from the other panels.

A single comic panel does not have to portray just a single instant in time as it can also represent an event that goes on for minutes.

Panel borders can have various functions. The basic function of panel borders is to simply limit a space in which objects and actions are drawn. Another use for borders is to not use them at all. This gives the frame the feeling of timelessness and unlimited space.

Figure 7. A panel without border to reinforce the idea of timelessness. In McCloud [12].

Inside the `panel` node we can find the nodes that describe the basic comics elements that make up a panel, namely, `background`, `character`, `narrative`, `balloon`, and `soundEffect`. In addition, the node `camera` is also defined at this level. Its purpose is to give information about the desired zoom-level and a target such as the face of a character or his pocket.

The `background` node defines the image that should be used to represent the panel's scenery. Inside it we can optionally define one or more objects that are part of the scenery through the `object` node and the respective images that visually depict them. Moreover it is also possible to define a location of the background where the object should be placed.

Backgrounds are used to situate the reader by depicting the place where the events are occurring. Another use for the backgrounds is to evoke an emotional or sensual response in the user. Figure 8 explores this situation.

Figure 8. Background as a means of transmitting emotion. In McCloud [12].

The `character` node defines the properties of the characters that will be present in each panel. The `emotion`, `pose` and `concern` nodes specify what feelings a character is expressing through facial expressions or body postures, what his pose is, and what his invisible concerns are respectively.

Invisible concerns are non-visual concepts that need to be shown in comics – the character's feelings and emotions. For instance, if a character has fallen in love, the artist can inform the reader about that by drawing little hearts around the character's head.

Figure 9. Portraying a character in love. In McCloud [12].

Further we can define what a character is doing through the `action` node and optionally what the `target` of his actions is. The character's `location` can also be defined optionally and the node `gaze` defines where the character is looking at. Like in the `background` node objects can also be defined inside the `character` node. The difference is that the object's location

refers to a place on the containing character so as to inform that the character is this object's owner.

If a given panel shall have narrative text the `narrative` node has to be used. This node makes it possible to define more than one narrative text for a single panel and optionally define a character or an object to which it is associated.

The `balloon` node specifies speech balloons. There are different types of balloons for different types of speech, namely, normal speech act, whisper, and thought, just to name a few. These balloons can then be combined to create melded balloons and connected balloons. Figure 10 exemplifies the mentioned balloon types.

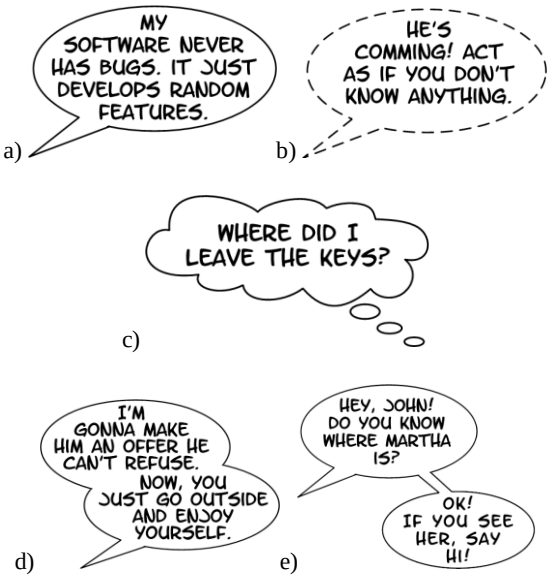

Figure 10. a) Normal speech act balloon. b) Whisper balloon. c) Thought balloon. d) Melded balloon. e) Connected balloon.

The different types of speech are chosen with the `type` attribute. A melded balloon is achieved by specifying more than one phrase where each `phrase` node is the text of one balloon. Since a single balloon can represent the speech of more than one character, it is possible to define more than one `owner` node, which correspond to existing characters of the actual panel. In case that this balloon needs to be connected to another balloon of the actual panel, the `connection` node can be used to indicate the target balloon.

A sound effect and optionally its origin can be specified by the `soundEffect` node.

Sound effects aren't actually real sounds, just a textual representation. So for example, if the artist wants to tell the reader that the phone in the image is ringing he can add the text "Riiiiiiing". It is also usual to represent these words in a big font size and with some text effects to give the sound a greater importance.

Figure 11. Textual representation of the sound produced by the cutting of a vegetable. In McCloud [12].

It is important to note that the Comic Strip Description Language follows a loose typing approach. This means that, for instance, the possible emotions' and objects' names are not pre-defined. An object can be any name as long as there is an object in the system's library with the same name.

The novelty of CSDL lies in its ability to describe comics in a way that it becomes possible to recreate them, at least to a certain degree. For instance, it is possible to specify which character or object was responsible for a certain sound effect. Or even what the current emotion and pose of a character is and where he is turned towards to.

4. THE COMICS2D SYSTEM

The Comics2D system aims at creating visual summaries for story-based applications with autonomous characters, such as Façade [11] or FearNot! [1]. It uses an automatic composition method.

The first component of the system is the CSDL markup language introduced above to describe the main events in the plot with a focus on the characters and on their actions.

The second component is a content library in which are stored all the images of the characters, objects and backgrounds that make up the comic strips.

The third and most important component is a plug-in based realizer which analyses and transforms the comic strip specification into a concrete visual representation.

4.1 Library

The library is where all the images that will be used in the comic are stored.

In the case of characters the necessary images are all the different facial expressions and body postures of each character. Separating the faces from the rest of the body allows for a more efficient reuse of a facial expression on a variety of body postures.

Objects on the other hand are single images. But they can also have multiple representations in the library to account for the different perspectives or positions in which the objects can be laid out.

The same principle applies for backgrounds. These images define the scenery in which the story takes place. To visually enrich the comic it is possible to store more than one image for the same scenery and so have slightly different perspectives.

Characters and backgrounds can have other information associated with them, namely, important positions on themselves. These positions are called "spots". A spot on a background could be, for instance, a place on a depicted kitchen table. On a character a spot could specify the position of a pocket. This way it is possible for the renderer to know where a book object can be placed and where the camera has to point at if the respective nodes in the CSDL require it so.

The library is stored in XML format. A database might seem the more appropriate choice for this specific application but requiring a database management system to use the Comics2D application could be too troublesome.

To ease the process of creating a library, a subsystem helps the insertion of images and the specification of spots.

It is possible to have multiple libraries. So, for instance, we can have a library with the characters drawn in western style and another library with the same characters drawn in manga style. This allows for the same story to be depicted in two different styles.

4.2 Realizer

The realizer is the heart of the system. It consists of a plug-in based application.

The application's main module has the responsibility of loading all the plug-ins and providing them an interface to access the library. After loading the plug-ins, the module loads the story description in Comic Strip Description Language format and feeds it to the plug-ins. This feeding process is done in seven phases, each one corresponding to the opening or closing of the key nodes of the CSDL (`comic`, `scene`, `panel`). So, for instance, in the first phase (`PreComicProcess`) every plug-in receives the full story description (the `comic` node and all its contents). The same goes for the last phase (`PostComicProcess`). The exception is the `PanelProcess` phase where the panel is actually produced. There the plug-ins receive the `panel` nodes of the comic story and produce the models of the visual representation of the story. Figure 12 shows the seven phases of the application loop and the CSDL nodes that trigger them.

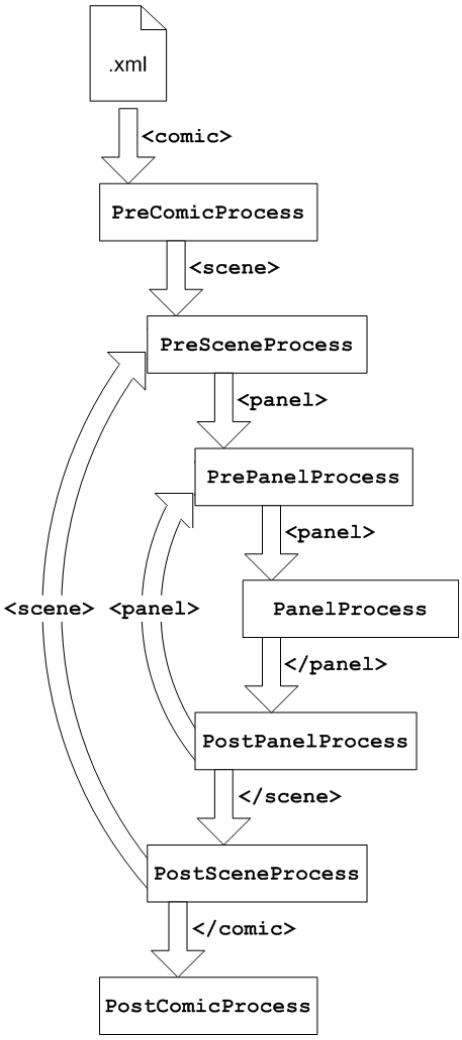

Figure 12. Main loop of the Comics2D application.

Every plug-in is designed for a specific task. Generate speech balloons and handle the sound effects are just two examples. These plug-ins are only active in the `PanelProcess` phase. An example of a plug-in that is active in the last phase is a layout algorithm. This kind of plug-ins can just work when every comic panel has been computed.

For a comic to be minimally acceptable it needs to have panels with backgrounds, characters and balloons. Additionally, the created panels need to be laid out. The application provides four base plug-ins to handle these tasks but they can be easily substituted by other plug-ins.

The base plug-in for the speech balloons implements advanced algorithms to make certain that the results are pleasing. So, for the construction of the balloon itself it implements Hurst et al.'s [8] algorithm for adaptive layout. This allows the text to adapt to the elliptic shape of the balloon. For the placement of the balloon it implements Chun et al.'s [4] algorithm. It places speech balloons according to algorithms used in automatic cartography. So, speech balloons are placed relative to the belonging character while maintaining the correct reading order.

The system specifies models for the internal representation of each element of the final comic. The internal representation could have been left to the plug-ins to decide but that would hinder a plug-in from accessing the model of another plug-in as there would be no standardized models. And this is of vital importance in this application. For instance, the speech balloon plug-ins need to access the model of the character plug-ins to know where the characters have been placed in order to place the speech balloons accordingly.

When requested, every plug-in has to export its models of the various panels in a specific format. One example is SVG [14]. This usually happens at the end, when the layout plug-in requests all the created comics elements. This approach makes it easy for a comic to be presented as a traditional comic in a paged layout or as a webcomic on an infinite canvas.

4.3 Advantages and Disadvantages

As mentioned, Comics2D uses the automatic composition method. This choice has some advantages and disadvantages that we summarise below.

- **Independence**: Using the composition method it is possible to have an independent system. The only requirement is to receive an input specifying the story of the comic strip. So, even if the system is made on purpose for a specific system where a story occurs, it can still be used as a standalone comics generator, as long as the input accords to a certain specification.
- **Modification**: The chosen method makes it easy to alter or update the pre-drawn objects if they are stored separately from the rest of the system. This has many different advantages. For instance, one might want to represent the same story with western style and Japanese style characters and can do so by only choosing a different image library.
- **Camera placement**: Following the composition method, the objects are, so to say, placed in front of the camera. This means that the camera placement is relatively easy needing only a few adjustments when zooming in or out, or focusing on a particular part of the scene. On the other hand, using the automatic transformation of dynamic graphics method the camera itself would have to be placed in the virtual world. Camera placement systems are not trivial, thus, having to implement one would be a disadvantage. However, if the system was well implemented the gain in shot perspective variety would be a big plus, allowing for more elaborate camera shots and more realism.
- **Visual appeal**: Even if the visual appeal of the comic depends largely on the correct layout of the characters and objects, the key to produce stunning comic strips is the drawing quality of the characters and objects that will compose the scenes. While computer graphics get better and better all the time, there is simply no comparison between professional human-drawn figures and stylized computer graphics. So, unless the drawings are of low quality, the composition method has an advantage over the screenshot method. However, having to pre-draw the elements is a disadvantage in itself. If the end result is to look good, a skilled artist has to be involved.

5. CONCLUSIONS AND FUTURE WORK

This paper presents Comics2D, a system to generate comics from story-based applications with autonomous characters.

We introduce a new markup language, the Comic Strip Description Language, which semantically describes a comic strip. Comparing CSDL to existing comics description languages shows that CSDL fills an important gap. While the existing description languages focus on describing comics that already exist, CSDL is thought as an approach to create new comics and therefore includes character-centred information such as emotions, poses and actions that help creating the story. We have done several tests to assess the viability of CSDL and came to a favourable conclusion. Our description language is capable of describing feature rich comics with many different styles. However, CSDL does not preserve layouts since its aim is to create new comics and not describe existing ones.

We also propose an application architecture that uses CSDL descriptions and transforms them in visual representations. The plug-in based architecture allows implementing different techniques of every aspect of comics. For instance, a layout plug-in can produce the resulting comic in a page-based layout and another layout plug-in can output a webcomic ready to be published on the Internet.

The Comics2D application is a work in progress. Our goal is to implement the full CSDL specification but we are currently focused on the most important elements, namely panels, characters and speech balloons since they are the basic constituents of a comic.

We believe that comics have the power to take story-based summarization techniques to the next level.

6. ACKNOWLEDGMENTS

This work is partially funded by the eCIRCUS project (contract no. IST-4-027656-STP) and by HUMAINE (contract no. 507422) both part of the Framework VI Programme from the European Community. The authors are solely responsible for the content of this publication. It does not represent the opinion of the EC, and the EC is not responsible for any use that might be made of data appearing therein.

7. REFERENCES

[1] Aylett, R., Louchart, S., Dias, J., Paiva, A., Vala, M., Woods, S., and Hall, L. Unscripted Narrative for Affectively Driven Characters. In *IEEE Computer Graphics and Applications 26*, 3, 2006, 42-52.

[2] Barzilay, R., and Elhadad, M. Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, 1997, 10-17.

[3] Cavazza, M., Charles, F., and Mead, S. J. Interactive Storytelling: From AI Experiment to New Media. In *Proceedings of the second international conference on Entertainment computing*, 2003, 1-8.

[4] Chun, B., Ryu, D., Hwang, W., and Cho, H. An Automated Procedure for Word Balloon Placement in Cinema Comics. ISVC, 2, 2006, 576-585.

[5] Comic Book Markup Language (CBML). www.cbml.org (in 2007-03-30)

[6] Comics Markup Language (ComicsML). http://comicsml.jmac.org (in 2007-03-30)
[7] Eisner, W. Comics & Sequential Art. Poorhouse Press, Tamarac, FL, 1985.
[8] Hurst, N., Marriott, K., and Moulder, P. Minimum sized text containment shapes. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, ACM Press, New York, NY, 2006, 3-12.
[9] Indigo Prophecy. www.atari.com/indigo (in 2007-03-30)
[10] Kurlander, D., Skelly, T., and Salesin, D. Comic Chat. *Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96*. ACM Press, New York, NY, 1996, 225-236.
[11] Mateas, M., and Stern, A. Façade: An Experiment in Building a Fully-Realized Interactive Drama. Game Developer's Conference: Game Design Track, 2003.
[12] McCloud, S. Understanding Comics. Kitchen Sink Press. Northampton, MA, 1993.
[13] McKeown, K., and Radev, D. Generating summaries of multiple news articles. In *Proceedings, 18th Annual Internacional ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995, 74-82.
[14] Scalable Vector Graphics 1.1 Specification. www.w3.org/TR/SVG (in 2007-03-30)
[15] Shamir, A., Rubinstein, M., and Levinboim, T. Generating Comics From 3D Interactive Computer Graphics. *IEEE Computer Graphics and Applications*, Volume 26, Number 3, 2006, 53-61.

Create a Comic Character: Making a Character Sheet in Adobe Illustrator. Make an Establishing Shot Using the Perspective Grid Tool in Adobe Illustrator. What You'll Be Creating.Â Working in Adobe Illustrator we can create a fast and flexible workflow by using clipping masks to create our comic panels, and the Effect menu to make wavy panels in no time. Find more incredible Comic Book Resources on Envato Market and Envato Elements. 1. How to Create a Storyboard. When you have a character and an idea of what you want to tell, it might be tempting to just dive in and start drawing. If you take extra care planning the story and the structure of the comic, you will avoid drawing yourself into a dead end. Step 1. Every comic needs a character. Characters help ground your story and make it interesting. A dynamic and entertaining protagonist is what will ultimately help your story sell. Once you know how to create a...Â There are some gag-a-day comics, such as comics published in local newspapers, but there are also comics that tend to be more serious. Many web comics have complicated and long running story lines with somewhat more complex characters. If you're looking for a simple format, consider the talking animal route and emulate comics like Garfield.Â If you're writing a genre-based comic, like fantasy, consider archetypes, which are stock characters that tend appear repeatedly in fiction. For example, an archetypal mentor would be wise, patient, and calm. Part 2.