Generating Comics Narrative to Summarize Wearable Computer Data

Jason Alderman

Masters Project Documentation

Information Design and Technology Program
Department of Literature, Communication and Culture
Georgia Institute of Technology

April 12, 2006

Committee:
Michael Mateas (Chair), Michael Nitsche, Thad Starner

Abstract

As people record their entire lives to disk, they need ways of summarizing and making sense of all of this data. Comics (and visual language) are a largely untapped medium for summarization, as they are already subtractive and abstract by nature (the brain fills in the blanks and the details), and they provide a way to present a series of everyday events as a memorable narrative that is easily skimmed. This research builds upon the research of Microsoft, FX Palo Alto Labs, ATR Labs, and others to further ground the procedural generation in the comics theory of Scott McCloud, et al.

Table of Contents

Introduction

In today's information-saturated world, context is everything. How do you take large tracts of raw data and imbue it with semantic meaning? The answer is rarely simple or easy. As information and interaction designers, it is our job to recognize patterns in information and make it understandable and accessible to all people.

Computers are becoming a larger and larger part of people's lives (Intel 2005). As prices for consumer electronics drop, more and more people are documenting their lives with digital images, video, music, and recordings (Imation 2003). Some researchers see the field of wearable computers (or wearables, for short) as the future of this trend. Wearable computers are small, unobtrusive, fully-featured computers that are, as their name implies, worn on a daily basis (Mann 1998). Wearables can be integrated with an array of sensors—cameras, microphones, etc.—to provide experience recording and memory assistance that extends even further (than standard digital camera and personal computer use) into the users' daily lives.

As storage space gets cheaper, and people record more and more and more, they need some way to automate the categorization of all of these digital memories (Lewis 2003). This is part of the impetus behind the Defense Advanced Research Projects Agency's Advanced Soldier Sensor Information System and Technology (DARPA's ASSIST) project, scaled-down successor to the LifeLog project (Schactman 2004). By outfitting troops on the ground—e.g., on patrol in Baghdad—with wearable computers and an array of sensors, and then databasing and cross-referencing the datastreams from these cameras, microphones, Global Positioning System (GPS) units, etc., ASSIST will enable these troops, as well as their commanders and intelligence analysts, to store and recall almost anything the troops encounter. It becomes an invaluable digital memory aid, provided that the information is sorted, indexed, and contextualized properly.

At present there are three ways that information such as this can be organized:

1. by taxonomy imposed by the individual at data capture (or sometime afterward, but in both cases, this is rather time consuming),
2. by "folksonomy," letting others in a community either classify or suggest the classification of your data (such as in social tagging sites like del.icio.us or Flickr), or
3. by categorization/indexing by the computer, based on characteristics such as textual content, metadata (data about the data itself, such as timestamps and author data), and any other kind of semantic data that can be parsed or calculated.

It is hard to ask computers to wade into semantics, but they *are* designed to do well things that we humans don't do well (or quickly): crunch through hundreds of data files, follow rules and scripts to do basic contextual analysis, and present these files in a way that allows people to infer deeper context more easily. Folksonomies are getting considerable attention nowadays, but they still have several drawbacks:

- Items can be tagged with several words that mean the same thing (a condition known as "**no synonym control**," common particularly between

different languages) but the computer will not realize that they mean the same thing; cellular phones are called "wireless" in the US, "mobile" in the UK, "handy" auf Deutsch, etc.

- Items can be tagged with **vague terms that have multiple meanings**; "hot" could mean high in temperature, chemically spicy, physically attractive, or selling very quickly.
- There may be privacy issues with certain objects that are tagged, in which the data should not or cannot be shared with others to be organized (as might be the case with certain classified government information).
- Folksonomies often depend upon people to do **tedious elementary analysis** that could be automated.

(Smith 2004) While this approach still might work for academic and civilian wearable computer users, it is not necessarily practical for soldiers, particularly, as mentioned, if the information in the data streams is in any way classified (in the security clearance sense of the word).

Yet even when this data is initially processed for context, wearable computer users need a way to visualize their past experiences—a format that is legible and quickly understood. I propose comics as that format. Most people in the US, Europe, and Asia have grown up with comics or some form of visual language and have no problem reading it and gathering meaning quickly; the facile nature of comics has led to their use in many school reading programs (Krashen 1996, Dotinga 2005); and simple picture-language icons have been used since at least the mid-1920s as a universal language in international venues such as airports (Neurath 1946). Comics, and their larger rubric of *visual language* (Cohn 2003, Horn 1998), inherit from a long history of speaking in pictures, dating back to Mayan and Egyptian hieroglyphs (McCloud 1993). By combining pictures, shapes, and words, comics can cleanly tell stories that are easily skimmed and very approachable. Several projects, which will be discussed below, have tried to capitalize on this medium as a visualization and communication tool.

Statement of Problem

Dr. Thad Starner's Contextual Computing Group (CCG) here at Georgia Tech has been working with researchers at the MIT Media Lab and IBM on a wearable solution for DARPA's ASSIST contract. This team is currently working on a map-based, spatial interface for the data they collect from their wearable computer ensemble, but they are also interested in a narrative-based visual summary that could let soldiers review months of historical data at a glance, helping them to annotate their memories of events shortly after the fact, or jogging their memories for passing on information in changeover debriefings at the end of their tours-of-duty. This design document lays the groundwork for the development of such a summarization tool.

I should make clear here and stress later that this tool is simply a summarization tool to aid the memory of end-users. While it may be a good visualization aid for explanations when the end-user is communicating with intelligence analysts or other personnel, it is **not** intended as an intelligence analysis tool, and it should not be used as such. (See further discussion of this issue in *Design Issues and Future Work* on page 31.)

This proposed software should take in time-stamped sensor data, initial information (metadata about the wearables), and preliminary contextual data computed by the CCG, and automatically generate a comic. Such software is not completely without precedent. Prior work in the field has made great strides, but it has been distributed: several research projects have each put in extensive efforts to apply portions of comics theory, but these parts have yet to be fully synthesized. This paper outlines the implementation methodology and theory that could make generated comics a more effective means of communication.

Wider Context

Why Narrative?

The rest of this paper will look at the question, "why comics?" First, let us ask, "why narrative?"

In short, people like stories. Or, put more accurately, people *remember* stories.

Roger Schank is a leading AI theorist and educational theorist who argues that human brains are wired for stories, that we think in stories (Kay 1995)—that memory itself is, in fact, mostly a collection of stories, and the process of telling stories is how we often create memories (Schank 1990). When we receive new information, we try to map it to stories that we already know that have similar goals and conditions. If we find a similar story to map it to, and there are details that do not fit this story, that excess information will be thrown out, cleaning the brain of information perceived as unnecessary (Verbarg 2006, Heuer 1999).

A study done with novice and experienced chess players helps bear out this story-memory theory; when asked to memorize pieces on a chess board, the experienced players were much more successful in remembering the locations of the pieces if they were from an actual game (even one they had never seen!), rather than randomly placed (Heuer 1999). They could remember these locations if they fit with episodic memories of actual chess games that they, as longtime chess players, had experienced before.

Therefore, by presenting these comics summarizations in narrative form, we should be creating summaries that are far more memorable than a simple presentation of events. By scaffolding the sensemaking process with suggested story, this software should help the user spend his time more productively, adding details and annotations that further embellish the summary artifact, allowing the user to refine his own memories of events, and enabling recipients of the summary to absorb the information from the wearable user much more quickly, as it will be couched in narrative. (There are strengths and weaknesses to this approach, which I address in the *Narrative vs. Record* section below in *Design Choices*.)

A (Very) Brief History of Comics Grammar and Theory

Fundamentals

In 1993, Scott McCloud published *Understanding Comics*, a 215-page treatise on the art theory of comics, told in comics form (McCloud 1993). McCloud's book built upon Mort Walker's *Lexicon of Comicana* (Walker 1980), which creatively named and identified the elements of cartoon art, and Will Eisner's *Comics & Sequential Art* (Eisner 1985), which began to codify the structure of comics as a storytelling medium. McCloud starts by defining comics as "juxtaposed pictorial and other images in deliberate sequence, intended to convey information and/or to produce an aesthetic response in the viewer," a definition which has been argued by many and somewhat tweaked by McCloud, but generally holds true. In highlighting the sequential nature of comics, he makes sure to establish a clear connection between the time and its representation as space on the comics page. (In *Reinventing Comics*, the follow-on, he also states that, like maps, comics can represent space in the world as space on the page, too (Horrocks 2003).)

The vocabulary of comics is made of visual icons and symbols that abstract reality. Abstraction, particularly the placement of simplified characters within a detailed world, helps people identify with the representations of people (avatars) in the comics.

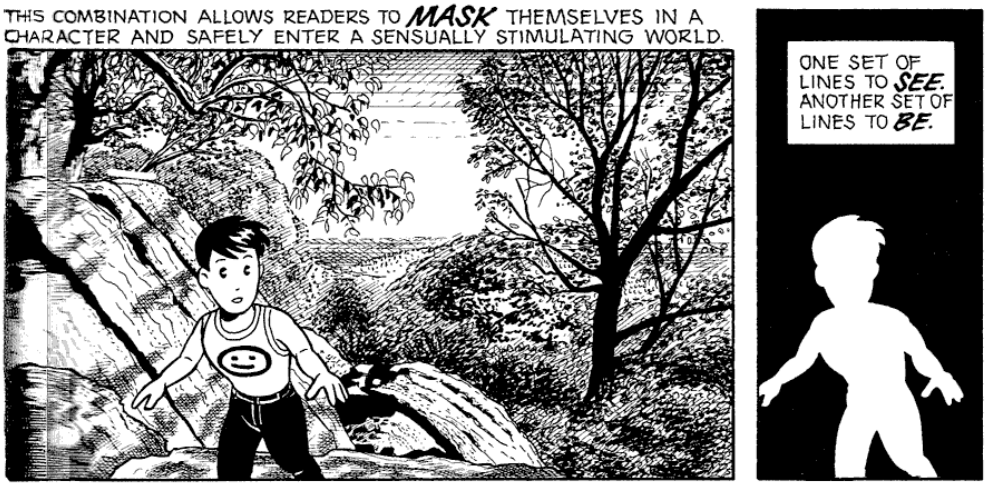


Fig. 1: Simplified characters + detailed backgrounds = reader immersion and identification. (p. 43, McCloud 1993)

McCloud fingers the gutter—the space between panels—as the place in comics where the action is. The difference in states between two comics panels calls upon a reader's imagination to fill in the blanks, creating the illusion of action or change (*closure*). Thus, a long, detailed narrative can be broken into tiny discrete chunks, like frames of film; these tiny chunks, as panels, can be selectively deleted, maintaining (in film terminology again) keyframes that depict the major changes; and the comics creator can rest upon certain cultural assumptions to expect that the reader puts together the whole story in her head without needing to be explicitly told all of the

minutia. The excerpt below (page 84-85) from *Understanding Comics* illustrates this subtractive nature of comics.

Closure, he states, is the grammar of comics (complementing the visual icon vocabulary), and to illustrate this statement, he identifies six different types of transitions between panels: moment-to-moment, action-to action, subject-to-subject, scene-to-scene, aspect-to-aspect, and non-sequitur. He also categorizes seven ways that words and pictures can work together on a page, stressing that the most effective comics use both together without letting the words be redundant with the pictures, or vice versa, in the expression of the message.

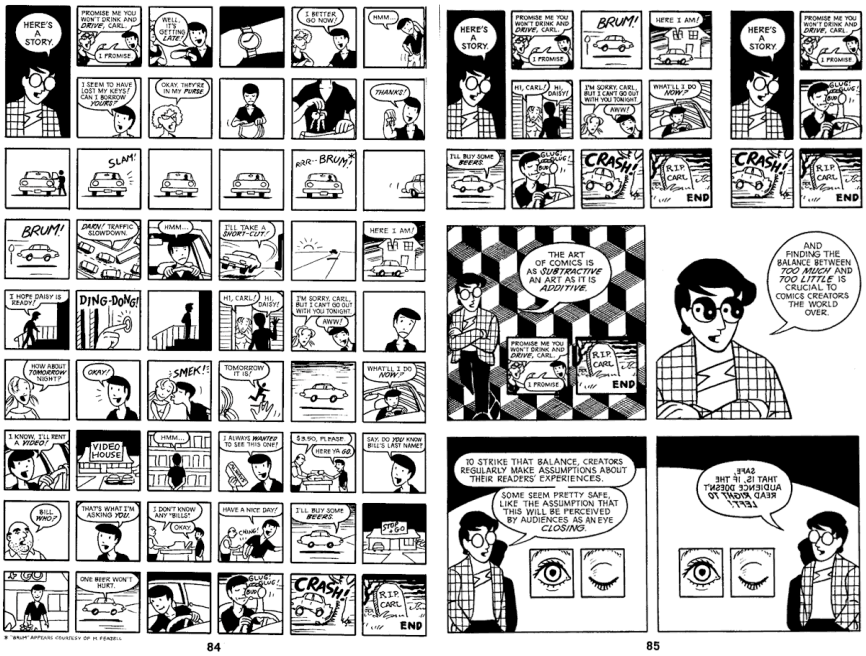


Fig. 2: Scott McCloud on the subtractive nature of comics. (p. 84-85, McCloud 1993)

Visual Language

Drawing inspiration from Scott McCloud, Robert Horn, known for his work in hypertext and information mapping, published a book in 1998 on what he called Visual Language. Horn's approach (also explained in visual-narration style using what appears to be Microsoft Office clip-art) suggests that in addition to pictures and prose, shape is also an important factor (Horn 1998). By citing shape as a pillar of his visual language, Horn places layout—both page layout and panel layout—as an important part of visual communication. (This is something that graphic designers will not argue.)

Neil Cohn takes a slightly different approach, weaving in the teachings of linguistic theorist Noam Chomsky. (Likewise, Cohn's use of the term *visual language* focuses far more on its *language* nature; both he and Horn appropriate the term for themselves, inadvertently colonizing a term already used in graphic design (Barber 2002).) Cohn dissects comics as a subset of visual language, and tries to break the medium down via the methods of linguistic analysis (Cohn 2003 & 2005). He states that McCloud's panel-to-panel transitions are insufficient (as are other mechanics of the McCloud definition of comics), and proposes that a generative structure similar to Chomskian hierarchies should be used instead of McCloud's mechanics. Just as

Chomsky talked of Noun Phrases, Verb Phrases, and Preposition Phrases, Cohn invents a hierarchy of Visual Syntactic Structures (VSS):

- Temporal Phrases (TP)
- Visual Phrases (WP, so as not to be confused with Verb Phrases)
- Environmental Phrases (EP)

He also establishes production rules for the combination of these TPs, WPs, and EPs, but these rules are more effective for analysis rather than generation. They do not work as well in reverse; a comic with the same exact panels can sometimes be generated by combining different yet plausible sets of phrases (although to Cohn's credit, he says that this part of the VSS theory needs work). To build these VSS phrases, Cohn calls upon a lexicon in which panels are attention units of different levels (micro, mono, macro, or polymorphic, as shown at right), and may be composed of smaller elements that are *productive* (representational objects, like people) or *conventional* (symbols such as speed lines, hearts, and word balloons)…ultimately very similar to the vocabulary set forth by McCloud and Walker.

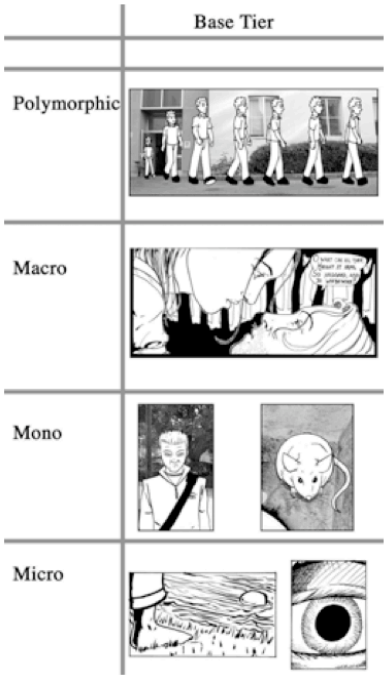


Fig. 3: Cohn's four levels of attention units [comics panels] (Cohn 2005d).

Through the Eyes of Graphic Design

John Barber, a webcomics artist hailing from Britain, draws upon Cohn's work in his MFA thesis in typo/graphic studies (Barber 2002). Barber argues that the amount of information needed in visual symbols to convey a clear meaning is even less than Cohn proposes (e.g., a silhouetted profile of a person running, in mid-stride, does not need the conventional symbol of "speed lines" to convey that the person is moving). However, the primary focus of Barber's paper is on the use of layout in comics as a narrative device. He experiments with several arrangements, ultimately breaking down each "page" on the screen into a 5x6 grid (or a 10x6 grid for a two-page spread—a 5:3 width-to-height ratio more conducive to the 4:3 screen than page- or square-shaped layouts), and he bases panel size proportions on this grid.

Barber also argues that comics operate on a dialectic of thesis-antithesis-synthesis that allows even single panels, properly composed, to be considered as comics. Even though there may not be sequence of images, the words and the images form a sequence that tells a story.

Comics Rhythm

In public talks, Scott McCloud often brings up the increasing importance of frequency modulation in comics (McCloud 2003). Most

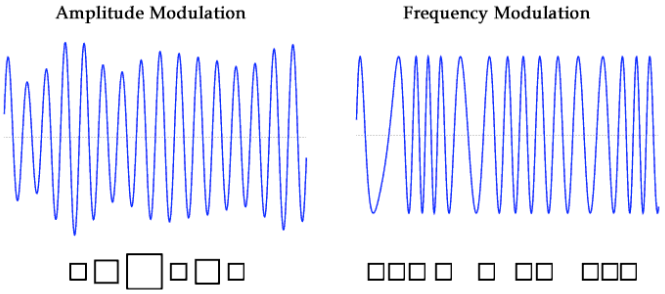


Fig. 4: Scott McCloud's father was a rocket scientist, which is why he says he feels compelled to use engineering terms (McCloud 2003). (Waveform images from http://en.wikipedia.org.)

comics, he levies, will vary their panels' amplitude (the panel size), but not their panels' frequency (the space between panels, or the gutter size).

For the most part, this is right. Printed comics have traditionally avoided white space with a fervor because added white space means less content (less story) on the small number of pages compared to thick paperbacks of Japanese *manga* (Barber 2002), and US comics consumers already purchasing relatively pricey small comic books have a tendency to balk at this practice.

McCloud likes to evangelize what he calls the "infinite canvas," the practice of making digital comics on an endless plane, using the computer's screen as a window to see only a small section of this vast canvas. Harkening back to such long historical "comics" as Egyptian wall paintings, Trajan's Column, and the Bayeaux Tapestry (McCloud 2000), many modern webcomics are composed in long vertical or horizontal scrolls that vary not only the panel size (as many comics do) but also the space between panels (known as the gutter).

Barber's thesis concurs with McCloud, reinforcing that the frequency of comics panels on the printed page is often bound to the size of the page and that chunking panels into stanzas of a set size (a two-page spread, what Barber refers to as "mise-en-page") creates a rhythm not unlike poetry.

> *The rhythm of the page is not a fixed beat, it is simply a beat that exists. Eisenstein suggests that there is a principle from which "the whole charm of poetry derives. Its rhythm arises as a conflict between the metric measure employed and the distribution of accents, overriding this measure."* [(Eisenstein 1949, p.48)] *The same can be said of a print comic's individual pacing and the inherent page structure.* (Barber 2002)

Once the constraints of the page are removed, it is up to the comics creator to create his own rhythm by varying the frequency on his own. McCloud is fond of showing the following webcomic by Canadian artist Jason Turner as an example of using the gutter size to modulate frequency.

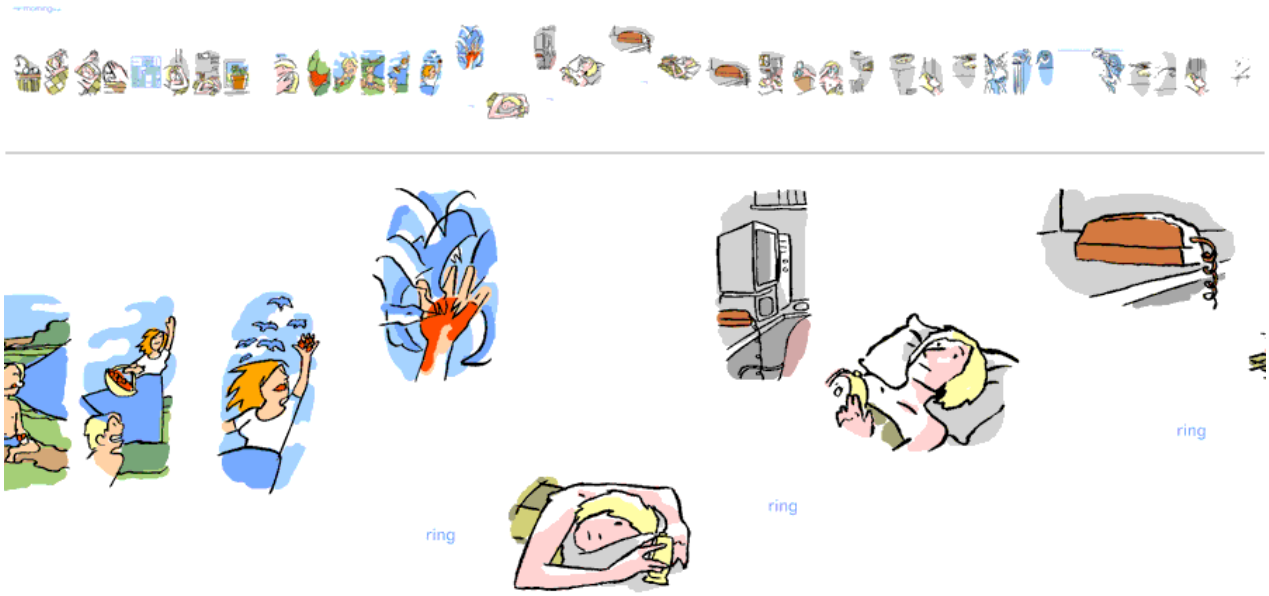


Fig. 5: Examples from Jason Turner's *Bright Morning Blue* (2002) of variations in gutter distances that change the rhythm of a scrolling comic. Top: A zoomed-out view of the comic. Bottom: Detail of frequency modulation.

While not making such an overt comparison to poetry, Eisner makes similar references to rhythm, in both the size and distribution of panels on a page (Eisner 1985), as well as in the *content* of the panels (Eisner 1996). Eisner argues that internal panel layout—choosing what is shown inside a panel's frame—is just as important for narrative rhythm. Film tends to leave much less to the imagination of the audience, Eisner asserts, and thus it will frame far more close-ups than are necessary to tell the same narrative in comics form. Comics have the luxury of frugality, in that they can tell much more story by selecting just the right images to summarize the narrative. Many recent comics artists, particularly in the 1990s (*Fig. 7*), tried to adopt cinematic language to their comics with varying effectiveness, but often their overuse of close-ups, Eisner would say, tended to hurt the rhythm of the story on the printed page.

Fig. 6. Comparison of film storytelling and comics storytelling from *Graphic Storytelling* (Eisner 1996, p. 73)

As pointed out by McCloud: in comics, less is more.

Fig. 7: Example of a comic emulating cinematic language with resulting unclear storytelling, by Rob Liefeld, from *Berzerkers #3*, http://www.art4comics.com/bezerk3.jpg

Story Grammars and Generation

Just as theorists have dissected the comics medium, so have scholars analyzed the narrative form. One such scholar was the Russian Formalist Vladimir Propp, who in 1928 published *Morphology of the Folk Tale* (Wikipedia 2004), breaking down the stories of Russian folk tales into discrete conceptual chunks (called morphemes) tied together with rules (grammar). Decades later, enterprising college students used his analysis in reverse, generating original folk tale stories from the same basic morphemes using the story grammar that he specified (Lim 2001).

By treating comics panels (Cohn's attention units) as the morphemes in the structure of a comic (rather than a story), one could also recombine panels and elements of panels to generate new comics by following the proper grammar. Artist and mathematician Jason Shiga has experimented with this approach in many of his print comics, producing works such as *Every Dog Has His Day*, with panels that can be arranged in several different patterns to tell different stories.

> *Imagine if you will, 16 unbound cards with four slots cut into the top of each one. Remarkably, you can start with the cards in any order and by performing four simple operations on the slots, order the cards perfectly. By performing the operations in reverse order, you get a second story recycling the same cards in a different order. The first story tells of a woman's experience with a disobediant [sic] dog and a romance with her boyfriend. The second story tells of the same woman's disobediant [sic] boyfriend and her love affair with the dog!*

Fig 8. Sample card.

(Shiga 2003). The stories, in these cases, exist in the mind of the reader, generated by the closure between the panels; when their order and context is changed, so does their interpretation. Shiga has also experimented with panel palindromes and other comics that play with form. His work in paper comics shows that story generation and comics are a well-suited pair.

Story grammars are, of course, not the only method of narrative generation, and as the section above on Cohn's visual language grammars mentions, grammatical deconstruction and reconstruction of comics is not the simplest or most effective approach. This project uses instead a scripted method that models the way that human authors would create the summary comics. Schank describes story creation as a five-step process of distillation, combination, elaboration, creation, and captioning. (Adaptation, a sixth step is not as relative here.) *Distillation* takes events and distills them into gists (story parts as memory structures) and then translates them back into language when needed. *Combination* assembles these gists into basic stories, suppressing story parts that do not support the message of the story. *Elaboration* can occur in three ways: detail addition (adding to the description of the storyworld often for the sake of extending the time taken to tell a story), commentary (elaborating on the storyteller's thoughts), or role-playing (elaborating on the thoughts of others). *Creation* is essentially choosing the style in which to tell the story to elicit the intended emotional response in the listener. Finally, *Captioning* is the practice of shortening a story by alluding to another story without telling it (Schank 1990). (Details on how this five-

stage process is used will be presented in the *Design Implementation* section below.) All in all, this process of analysis of activities, correlation to existing story templates, and fusion into a new story provides a good general framework for the generation of comics narrative.

Survey of Comics Generation Tools

Computer-based comics generation tools run the gamut:

1. Tools that generate comics automatically from an interaction corpus, i.e., the body of raw data produced by a person's interactions with computers
2. Tools that generate comics randomly by recombining and/or procedurally generating with no input from a user
3. Tools that let users make their own comics, by assisting the user with clip-art and comics components that they can recombine as they see fit

Tools of the third type naturally offer the most agency in creating panels and layouts of any size and shape. In tools of the first two types, the computer is charged with a greater role in the creation of the comic, so the tools generally keep panel sizes and shapes uniform; the technical issues of storytelling within panels provide enough of a challenge to designers that generating layout as well adds considerable complexity.

Appendix 1 shows a survey of comic generation tools. Of these programs, the closest project to this endeavor is ComicDiary, from Tokyo's ATR Lab, which takes data from a conference-goer or lab-visitor's experience (obtained through sensors on the person and in the rooms, and data from the person's PDA), parses it for context (figuring out the user's schedule, business card exchanges, locations at certain times, and surveyed favorite conference experiences), and generates a humorous story as a memento for the visitor (Sumi 2002). ComicDiary is also implemented as a Macromedia (now Adobe) Flash file driven by external text files generated by server-side scripts. The narratives generated by ComicDiary are exaggerated slightly to make the stories more engaging; this lack of specificity also lets the user fill in the details from his or her memory, rather than flagging the computer's inability to calculate the context of every situation. (The extremely simplified cartoons of ComicDiary are very helpful in letting the reader identify with their avatar (McCloud 1993), but this cartoonishness would most likely be *less* effective for more serious uses, such as the summarization of military activities.)

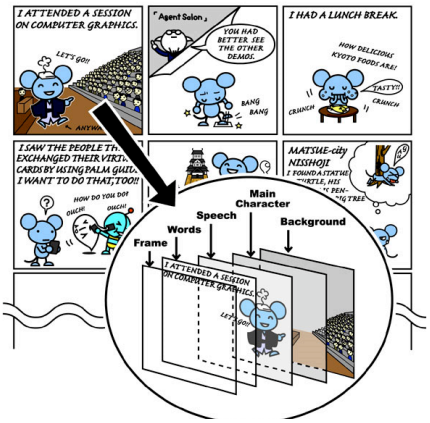


Fig. 9: A ComicDiary page, dissected.

ComicDiary builds upon Microsoft's Comic Chat, which also used a library of pre-drawn comic elements to generate comic panels that form a transcript of a conversation in an Internet Relay Chat channel (Kurlander, et al. 1996). Comic Chat parsed the text of the exchanges between participants for simple context, changing the artwork for the characters within the panels so that, e.g., a person who typed in ALL CAPITAL LETTERS would be represented by a shouting avatar. (The user's intent

might not have been to shout, but the shallow-context computation proved effective in most cases for the program.) In both the cases of ComicDiary and Comic Chat, leveraging layout of panels in different sizes and shapes was an item specified for future work.

However, of all of these tools, the only procedural comics tool that varies panel size is Video Manga (Uchihashi 1999, Uchihashi & Foote 1999, Boreczky 2000). Video Manga (VM) is an especially ambitious project, as it attempts to parse a video of a meeting into semantically sized comics panels using primarily the data within the video clip itself. VM segments video clips into keyframe clusters delineated using both video analysis and meeting minutes. It then ranks each segment with importance scores based on rarity and duration, and captures an image from the middle of each segment and scales it relative to its importance. These panels of video stills are then connected by faint line trails to show the reading order, packed together horizontally, and resized slightly to fit, all to improve readability and economize space when they are presented on a computer screen. Panels can be clicked to watch video clips or to expand a panel into constituent panels for finer time resolution. Captions, drawn from the time-stamped meeting minutes, are placed on panels in an unobtrusive pop-up nature, as early testing discovered that text on small panels was hard to read and obscured too much of the preview image. In user testing of the manga style of video summarization (panels chosen and sized based upon a semantic importance ranking), versus a selected summarization (same-size panels chosen from the importance ranking) and a control group (same-size panels of video stills collected at a constant interval with no importance ranking), "study participants judged the manga summary to be significantly better than [selected and control] with respect to their suitability for summaries and navigation, and their visual appeal" (Boreczky 2000).

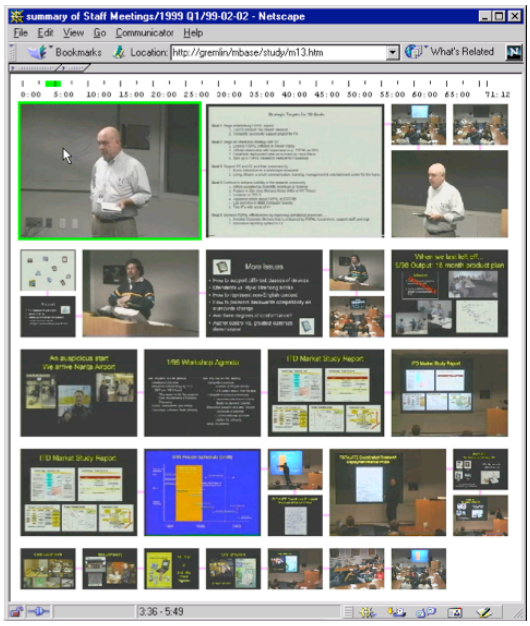


Fig. 10: FX Palo Alto Labs' Video Manga in action.

Video Manga is a tremendously impressive project with regards to emulating the style of comics in a procedurally generated work. FX PAL researchers even developed a free-form version of their generated comics that varies gutter size and external panel layout to to create a more natural reading rhythm (Uchihashi 1999). Future work on this project may explore speech recognition and speaker recognition (through face recognition and other methods), and use these technologies to produce even more comic-like output, with word balloons, filled with text, that correspond to appropriate speakers.

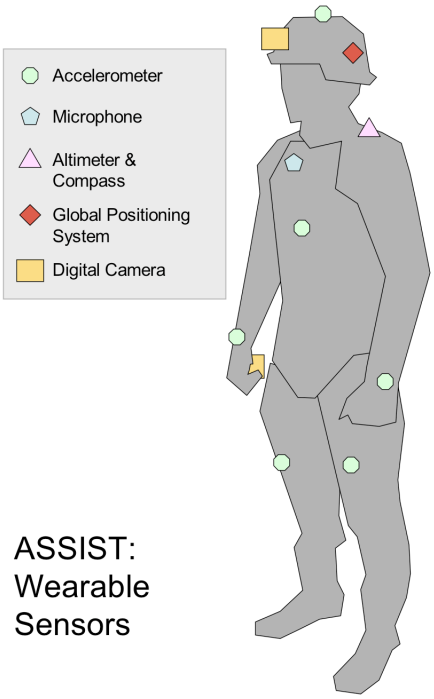


Fig. 11: Current configuration of ASSIST sensors (wearable computer is on back).

The ASSIST Project

As explained above, the ASSIST project aims to help soldiers remember details of important experiences in the field, and to help their commanders and intelligence support get additional datastreams to bolster after-action debriefings. In all, it should aid the situational awareness of the entire chain of command involved in field operations.

Each soldier will be fitted with an array of sensors as shown: accelerometers on the head, chest, wrists, weapon, hip, and right thigh; an altimeter and a digital compass to provide elevation and orientation; a microphone and digital cameras, and a global positioning system (GPS) receiver. The head camera is a small Canon point-and-shoot digital camera, and an additional digital camera or video camera may also be carried. These sensors are all connected to a small OQO computer stowed in a modified CamelBak water-pouch backpack.

The same suite of sensors could just as easily be used on a civilian (non-military) wearable computer user configuration. In both cases, the comics summarization of the sensor data, like ComicDiary, could provide a memory-jogging memento of a day's events, but in a way that might help improve the accuracy of reporting for situational awareness and intelligence gathering. Comics summaries could be annotated by the wearer after the fact to provide further context, and would also serve as an interface to the raw photographic and audio data collected by the soldier. Dr. Starner envisions such summaries as a way for deployed troops to send an easily-read and automatically-generated visual journal of daily life back to friends and family, as well as a way to summarize months of a year-long (or longer) deployment to pass on significant events to incoming replacements. Should ASSIST be widely deployed, and should the upward trend in milbloggers (military members with unofficial weblogs chronicling their daily lives) continue (Memmott 2005), such a feature could become tremendously popular and useful. Unfortunately, these are a lot of "shoulds," and current military public affairs policy would likely strike down the usage in blogs—at least among military personnel—before it begins, unless someone heavily sanitizes or otherwise edits the data to be summarized.

Design Implementation

Design Goals

Breaking Down the Problem

The problem posed (see *Statement of Problem* above) was to generate a comics narrative from a semantically-analyzed interaction corpus. On the surface, this may seem a straightforward problem, but to generate comics that are *comics*, and not simply a series of images placed on a grid—taking into account narrative flow, the affordances of the comics medium discussed above, and the skimmable nature of the final product—there are several steps that need to be taken.

1. Define an XML format for the semantically-tagged data that describes activities, locations, and directions.
2. Write a set of rules for finding patterns in these activities that match it to existing templates of stories and generate a narrative
3. Write a set of rules for ranking activities and narrative elements by semantic importance to determine which will become panels and what size these panels will be
4. Define an XML format for describing panel sizes and content
5. Generate an XML document describing the comics narrative from the initial XML document activity data
6. Write a set of rules for rendering internal panel layout (composition)
7. Write a set of rules for rendering external panel layout (shape, amplitude, frequency)
8. Generate comics from XML panel description from this data and pack the panels into the space available
9. Package it all in a functional user interface (UI) that allows the user to adjust the time scale and read the generated narrative summaries

Thus, the process behind the scenes for this software would be as follows:

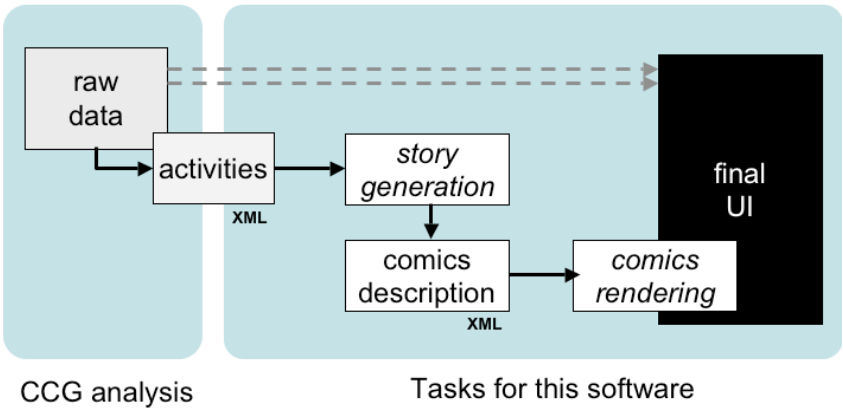


Fig. 12: Role for this summarization software and tasks it should perform.

Goals for Software

The ultimate goal for this software is to generate a comics summary of captured events that is browsable on multiple timescales (by minute to by month) and is wrapped to templates of known narratives. The software leverage the use of pictures, words, and shapes inherent in the comics medium, and should integrate the features that work well in previous works:

Software	(+) PRO Lessons Learned	(-) CON Lessons Learned
ComicDiary	Should generate memorable narrative from multiple datastreams and sources.	Should avoid fictionalizing narrative; should make comics abstracted and simple but not overly cartoony.
Comic Chat	Should balance placement of characters within panel to match activities in panel	Should avoid clip-art effect through more natural zoom, simpler art, and wider range of keyframes [or alternately (see future work in *Design Issues*) poseable avatars with greater range of motion]
Video Manga	Should vary the size and spacing of panels on the virtual page; use VM's panel-packing algorithm to make page read flow smooth.	Should vary panel width, height, and gutter to make panels look more organic; should vary border colors and shapes to draw attention to important panels.

Additional features, such as integration with GPS data to show location on a map, and the ability to browse media assets related to the time period of a certain panel are not considered core functionality; they may be added in future versions, and have been briefly outlined in the user interface section below.

Design Choices

Narrative vs. Report

This document mentions earlier (in *Why Narrative?* under *Wider Context*) that there are strengths and weaknesses to presenting a comics summarization in narrative form. The alternative would be to simply present a series of events, a record of the actions that occur, without trying to match these actions to a narrative script.

One of the problems with this approach is that it would be very difficult to summarize a record without knowing what the story behind it was. One could attempt to do statistical analysis on the activities detected, determine least common and longest duration activities in the run, and include only these activities, but these statistics would not make much sense if you had no context of how rare these activities were for

the type of mission that the soldier was assigned. If the soldier is on a presence patrol, for example, then spends the whole mission in a firefight, a statistical analysis of the events might assume that walking with one's weapon down was less common, and highlight these moments rather than the details of the firefight. The resulting comic would probably not make much sense. It would just be a series of statistically infrequent and unexpected activities, relative only to that run, placed together with little context. Without the context of a even little bit of narrative scaffolding (i.e., the mission type—whether "presence patrol," "cordon and search," etc.—has a set of goals and a storyline of typical activities), the sequence of panels generated as a summary is bound to confuse a reader.

However, Scott McCloud points out that people will find meaning in even the most random, non-sequitur sequences of images. "Such transitions [between even apparently totally unrelated images] may not make 'sense' in any traditional way, but still a relationship of some sort will inevitably develop" (McCloud 1993). If people make stories from a series of events, seeing patterns where there were none deliberately put, then why bother trying to inject narrative at all?

The stated goal of this project is to help users organize and distill their memories of events for posterity; by providing some context, narrative structure makes this possible. As pointed out, even the very act of summarization itself relies on such narrative context. Without narrative, it would *not* be a summary, but a report of events with arbitrary omissions made to shorten it. Thus a "pure" report is out of the question.

Furthermore, a "pure" narrative is similarly out of the question, for many reasons. The context required to independently compute, from a series of events, the exact story the wearable computer user sees is a virtually impossible task. Because narrative is ultimately highly dependent upon the point of view of both the storyteller and the audience (Schank 1990), the algorithm would need to model the psychology, goals and beliefs of both the storyteller and his listener, on top of creating a story that made sense and fully understood the context of all recorded activities. By example:

> *Perhaps running is not an atypical activity for a soldier on patrol, but in the case of our storyteller, she had seen a doctor earlier that day who recommended that she not run for medical reasons. This diagnosis would make a recorded running event significant. On the contrary, what if the doctor's diagnosis was overly cautious, in the perspective of the soldier? What if, among her peers, it was common practice to ignore this diagnosis for practical realities, and none of the platoon had yet seen any adverse side effects of ignoring doctors' orders? This detailed context, which might deflate the importance of running once again, is currently nearly impossible for a computer to assess.*

Should these factors simply be ignored, the software either produces a comics summarization that is still slightly report-like, or one that compromises actual events or conditions to tell a perfect, pre-defined story.

Thus, the summarization has to fall on the spectrum somewhere between pure report and pure narrative. Should it lean more toward report, or more toward narrative? I argue that it will be *slightly* more narrative, but will be balanced toward the middle. The summarizations are intended as a memory aid, so the software should not

intentionally create fictions that significantly alter the reporting of events. Any narrative generation will invent something to fill in blanks in the events to make a story, but the goal is let the events tell the story as much as possible. ComicDiary intentionally *does* create fictions because its purpose is to be entertaining. Our software's purpose is to organize and clarify memories, so it should try to mitigate these inevitable fictions as best it can. How, exactly?

As mentioned earlier, Schank talks about three types of elaboration in story creation: detail addition, commentary, and role-playing (1990). ComicDiary leans away from the report side of the spectrum by elaborating with scripted commentary (humorous remarks) and role-playing (explicitly remarking on the thoughts of the user and the people she meets). Detail addition is also tricky because, should details differ greatly between what the user recalls and the computer suggests, then the summarization could be ineffective. The user might reject the summarization outright, or create false memories of events inspired by these procedurally generated details. In essence, our summarization software should create a story that is detailed enough that it is memorable and not too confusing to follow (the non-sequitur effect), yet also vague enough that users do not get tripped up on detail discrepancies.

Therefore, detail addition by the computer can only occur effectively if the user knows about it and endorses it. Two ways to accomplish this agency are to make clear the story skeleton to which the activities are being mapped, allowing the user to change this story archetype to see how the change in narrative changes the summary, and to mark (through different border types, colors, or another method) panels that are generated in support of this chosen story structure. For example, if the narrative branch chosen is one in which a roadside bomb (IED, or Improvised Explosive Device) is discovered, then a small panel could be added earlier in the sequence showing a person setting the IED at the background location where the IED is later discovered in the comics narrative. This is a detail not explicitly stated in the activity data, but if the data points to the setting of an IED, it logically follows that someone put it there at some time in the past, and to improve the narrative flow, a panel of setup (or foreshadowing) should be placed earlier in the sequence of summary panels. Otherwise, the eventual detonation of an IED may seem jarring, for reasons not related to the emotional impact of the event itself.

Fig. 13: Pedestrian Crossing sign (top), work of Josh McKible, http://www.mckibillo.com/ (bottom)

Abstraction

Like the story, the pictures used in the comics summarization also need to be simple and abstract enough that the user can identify with the avatar in the panels, as described above in the *Wider Context* section, but not so detailed that the discrepancies

with the user's memories are jarring. For the purposes of this implementation, I have kept avatars as genderless characters that are only slightly more detailed than the figures on street crossing signs and warning labels. The comics summarizations could probably employ without issue more detailed, yet faceless, avatars, like the illustration work of Josh McKible (http://www.mckibillo.com) , but developing the art for such of characters would require significantly more time and effort.

Also, for the ASSIST project, where summaries are being generated for military personnel, it is important that the avatars not become more "cartoony" or exaggerated than a street crossing sign figure. This would compromise the seriousness of the events summarized; the shooting death of your close friend in combat would not be something that you would want visually depicted as a cute cartoon doodle. That presentation would completely sabotage the effectiveness of the summarization as it is rendered.

Why use Flash?

While Processing (http://processing.org) was also a contender, Flash is presently the best option for the implementation for its popularity as a development platform, its vector graphics, its excellent masking capabilities, and its built-in-support for XML.

Flash generates small binary .SWF (Shockwave Flash) files which bundle together vector graphics (images that are described mathematically, and therefore can be resized smoothly) with script code that is evaluated at runtime. Flash uses ActionScript, which is a derivation of ECMAScript, the standardized version of common JavaScript used on most web pages. ActionScript has been in use (and has been slowly refined) over the last four generations of the software, and has become a very widely-used scripting language, with copious documentation from both Adobe and from the developer community at large.

While not an open standard, its popularity has made it an almost-*de-facto* standard in cases of vector graphics. Flash runs on almost every computer platform, with players available for many modern cellular phones and other portable devices as well. Scalable Vector Graphics (SVG), an open XML standard for scriptable vector graphics is still not in wide use; the latest version of Mozilla Firefox (1.5+) supports only a limited subset of SVG's specified functionality. At present, SVG is not as commonly used (and thus supported) as Flash binaries.

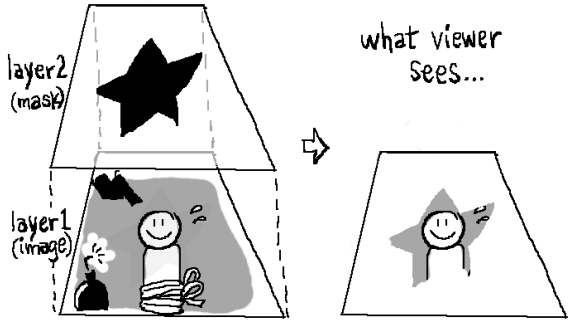


Fig. 14: Explanation of a clipping mask.

The deciding factor, however, was that Flash currently has the best support for clipping masks of any of the rapid-prototyping tools. Clipping masks are a way of using a foreground shape to define the shape of what is visible in underlying graphics; such functionality is extremely helpful in the creation of comics panels because it allows the computer to create a vector image which can be dynamically zoomed and then cropped to the shape of

any frame. To do this in Java would require writing a set of masking algorithms, which would undoubtedly take more time than learning ActionScript, and most likely would not allow for the variety of panel shapes that Flash's masking allows. Flash lets any MovieClip (the atomic object of a Flash file) be assigned dynamically as a clipping mask for any other MovieClip. MovieClips can be created on the fly from a script, set as masks from a script, and nested nearly infinitely, allowing for rather complicated layouts to be generated in relatively few lines of code.

In addition, recent versions of Flash have built-in support for parsing XML which makes it relatively simple to drive the creation of graphics in the final SWF with data from a text file or dynamic streaming data.

Implementation Methodology

XML Format for Activities

Raw data from the wearable sensors should be compiled into an XML file of the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<activities>
      <run start="2005-11-17-14.48.22">
            <activity type="walking" dura="00.04.00" />
            <activity type="walking" dura="00.02.00" />
            <activity type="standing" dura="01.01.32" />
            <activity type="walking" dura="00.05.00" />
            <activity type="kneeling" dura="00.04.00" />
            <activity type="standing" dura="00.02.34" />
            <activity type="shakinghands" dura="00.00.01" />
      </run>
</activities>
```

All activities are enclosed within a root `<activities>` element. Inside this element are one or more `<run>` elements with an associated start date and time, in the format of `YYYY-MM-DD.hh.mm.ss`. (This datetime format has been chosen arbitrarily because, at the time of this writing, it matched the file-naming convention of the ASSIST media captures from the sensors.) Each `<run>` contains an uninterrupted series of activities culled from the data, along with their associated durations in singleton `<activity />` elements. Duration in these elements is similarly in the format `hh.mm.ss`, although (as with the start time) it is acceptable if the seconds are in decimal format `hh.mm.ss.ssss`. Logically, there should be an activity of *some* type identified for every moment that the sensors are active; even if a wearer is not doing anything, they should at least be standing, sitting, kneeling, or lying down. Should there be a gap in the activities collected, for any reason (e.g., sensor malfunction or downtime), the `<run>` element will end and a new one will start when the activity reportage resumes.

Activity types are unique; a complete listing is given in Appendix 4. If a wearer is both kneeling and shaking hands, there is a specific `type` string for this combination of activities. The amount of activities examined in the current set is small enough that

this should not matter. Should concurrent activities be desired in separate listings a future document, the XML format will have to be modified.

Narrative Generation

In the prototype described in this document, the XML file that lists activities is parsed by a server-based PHP script into a string of single letters signifying activity types followed by an integer signifying the duration in seconds.

Currently this script compares activities to a set of expected activities in that sequence (e.g., in a soldier's presence patrol, walking, standing, and kneeling). If it comes across an activity that it has been told by its script, through a regular expression match, is unexpected (e.g., rifle raised suddenly), then a boolean flag is tripped. The PHP script now compares the unexpected activity finding to one of a few possible narrative branches (e.g., pursuit of suspect? sniper attack?) and looks at the subsequent activities to try to get the best match. Activities are currently arbitrarily weighted for rarity in a "presence patrol" type of mission; combined with duration, these rankings help determine the size of the panel that is ultimately generated. In a fully functional prototype of the software, this schema of rarity rankings/probabilities and activity branches would be one of several schemata, one for each different mission type, that could be interchanged by the wearable user to test the accuracy of the narrative generation.

A preferred way to do this analysis, however, would be with statistical grammar rather than a context-free one. This would presume that a large dataset could be analyzed for the statistical probabilities that a particular activity would follow the activity in question, given the activities that precede it; these probabilities would then be used to flag unexpected activities.

As the primary purpose of this software is as a memory aid, the overall process of narrative generation should loosely follow Schank's model for the creation of stories (and by association, memories) detailed in *Tell Me A Story* (1990). Schank expresses six steps in story creation: distillation, combination, elaboration, creation, captioning, and adaptation.

In distillation, story elements are first boiled down into "gists" which are the structures in memory in which they are kept. This is akin to our analysis of activities from the activities.xml file to sort into apropos story activities in an array. Next, these gists must be able to be translated from an abstract memory format (the gist) into English (or, in our case, comics panels).

These story parts are then combined and assembled, matching different story parts together to make a new composite, through the acts of conjunction and suppression. Aspects of stories that are combined that do not help to tell the story, but do not invalidate that story part from being used, are suppressed, while other aspects may be conjoined. This is the first step of summarization in our case, as our approach takes multiple similar activities (e.g., walking, then standing, then walking, then standing, then walking…) and averages them together, joining several of the walking activities and several of the standing activities each into one panel.

Once the basics of the story have been assembled in this relatively formless lump, the story creator must elaborate on these elements. Schank asserts there are three types of elaboration: detail addition, commentary, and role-playing. Our system avoids commentary (thoughts of the story-creator) except in after-the-fact annotation of panels, since the computer cannot fully guess the intentionality of the wearable computer user and such guessing would probably only make the summarization less real or less helpful. It also generally avoids role-playing, as Schank defines this elaboration as speculation on the thought processes of other actors. While the combination of networked data from other users together with one's own might occur in ways that make such role-playing fruitful (if only to make educated guesses on the thoughts behind the actions of fellow wearable users), this software must be cautious in drawing conclusions on the behaviors of other soldiers as if they were more-predictable software agents.

A story is actually created by matching these combinations and elaborations up to a story skeleton, like a set of magnetic poetry, to see what works. Each run is currently assumed to be its own narrative. The strategy of this software does not provide for subplots or superplots that may arch over several runs. If this is a feature that users need, it may be considered in the future, but for now, it is not treated as a requirement.

Schank talks about "captioning," which shortens the resulting story by referring to another embedded story in the story…essentially the act of self-summarization after the story has been created. In this case, the captioning should occur before the comic is actually put together and rendered; this is one advantage that our comics generation method has over other summarization methods, which usually start from the raw data, put together as much of it as possible, and then slice off unnecessary parts. Since the story does not exist in summarizable form until we construct it to begin with, we can construct it with an eye to striking out parts while making the story, so that the actual comic instantiation act does not create a full story which is then trimmed of fallow Panel objects.

Finally, Schank does not use "adaptation" often, but claims that it is a further translation of the final story into other media. The story that we create is adapted in this sense from its activity-event-narrative roots, into comics panels with a variety of properties to match the narrative. These comics panels are then represented in an XML format which is again adapted by Flash into actual comics.

XML Format for Comics and Panel Descriptions

The XML format for describing the comics is quite loosely based on the ComicsML format (McIntosh 2003). ComicsML is a XML description of a comic that is designed primarily for web-based serial comic strips (rather than longer form comics) with the intention of marking up a single-image comic in text form so that it can be indexed by search engines. It specifically avoids layout, and is very dialogue-centric; this project is layout-focused and avoids dialogue, but ComicsML still provides a relatively good framework from which to start.

A typical XML file describing a comic should look something like the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<comic>
  <title>Test comic</title>
  <last-built>2006-03-22-02.07.02</last-built>
  <description>Expository text about the comic in question.</description>
  <url>comic.xml</url>
  <panelsize-default>
    <width>2</width>
    <height>2</height>
    <gutter-before>0</gutter-before>
    <gutter-after>0</gutter-after>
      <border>
       <shape>normal</shape>
       <thickness>3</thickness>
       <color>0x000000</color>
      </border>
      <type>macro</type>
  </panelsize-default>
  <panels>
    <panel>
      <size>
        <general>wide</general>
      </size>
      <panel-desc>
        <action>
          <activity>walking</activity>
        </action>
        <composition>
          <type>macro</type>
        </composition>
      </panel-desc>
    </panel>
    <panel>
      <size>
        <general>big</general>
      </size>
      <border>
        <shape>blast</shape>
        <thickness>6</thickness>
        <color>0xff0000</color>
      </border>
      <gutter>
        <gutter-before>2</gutter-before>
        <gutter-after>1</gutter-after>
      </gutter>
      <panel-desc>
        <action>
          <activity object="door">standing_weapon</activity>
        </action>
        <composition>
          <type>macro</type>
        </composition>
      </panel-desc>
```

```
    </panel>
  </panels>
</comic>
```

As shown, the root element of the XML document is `<comic>`, with metadata for the comic, including `<panelsize-default>` specifications, at top, and the `<panel>` elements below inside an element called `<panels>`. Panels are laid out on an arbitrary square grid, 10 wide by 6 high (each grid square approximately 60 x 60 pixels), and units for default panel size and gutter size are specified in numbers of whole squares. (Video Manga uses an 8x8 square grid for panel layout (Uchihashi & Foote 1999); Barber's final design project "External Syntagm Variations" mentioned earlier uses a two-page spread with a 5x6 square grid on each page (2002); I borrowed from Barber because his mise-en-page layout—the 10x6 spread—was a better fit for the dimensions of the screen.) Using the default grid square size as a baseline, the typical comics panel will be just shy of 120 x 120 pixels, and the entire visible comic will be 600 x 360 pixels. Since we are working with scalable vector images, these proportions work quite well to fit onto a screen resolution of roughly 640 x 480, and scale upwards smoothly.

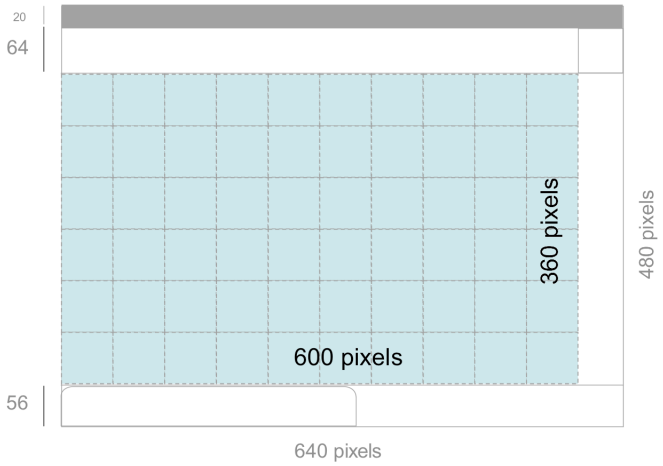


Fig. 15: The 10x6 grid fits on a 640x480 scale screen when its squares are 60x60 pixels. More details at Appendix #.

Panel locations, panel sizes, and gutter sizes, which will be discussed below in *Rules for External Panel Layout*, are expressed along this grid of squares. The comic description does not currently require coordinates on this grid (with 0,0 as the top left square) to place panels, but instead places panels relative to each other. The panel parameters "gutter-before" and "gutter-after" allow a great deal of flexibility in the rendering stage to determine how the panels can be placed on the virtual page.

Transitioning from external layout to internal layout: the `<panel-desc>` element describes what goes on inside the panel, and contains two important elements in `<action>` and `<composition>`. The action element describes the activity that is being conducted by the user's avatar in that panel, and optionally specifies whether or not there will be a subject character or prop in that panel as well (e.g., shaking hands will always have a subject, but a specific avatar could be selected; running after a suspect could just show the user's avatar running, or could show the user's avatar running and another avatar for the suspect also running, in front of the user's character.). Currently there is no distinction made for activities that are done *to* the user's avatar, such as if the user were being pursued.

The composition element specifies a visual attention unit level, a la Cohn (*see Fig. 3*), that essentially gives a zoom range for that panel. More information on how this panel parameter is set is discussed in the following section on internal panel layout.

Rules for Internal Panel Layout

Internal panel layout consists of the placement and framing of elements within the panel. Characters, props, background (setting), and symbology (word balloons and other *emanata* symbols (Walker 1980) that arise from characters, largely unused in this work for reasons detailed below) are all composed to communicate the action occurring to the reader.

The limitations of the current implementation simplify internal layout significantly. Currently each panel only has one focal point (the avatar of the wearer, or another singular subject of the panel), or, *at the very most*, two (the subject of the panel and the subject of its action). These focus points are kept within the panel, and the panel centers on it in the case of a single point, or on a point equidistant between them both in the case of two points.

Zoom level should generally start out at a distance for all activities, to show the full body of the subject (somewhat akin to Cohn's "macro" level for panels), but as the activity is repeated, the panel frames a closer and closer view of the subject (Cohn's "mono" level) until a rarer (unexpected) activity interrupts the cycle. Due to the lack of detail in the characters, this prototype avoids extreme close-ups (Cohn's "micro" level). Without more drawn detail, such imagery would more likely be confusing to the reader. (Is that a stubby rectangle… A hand? A gun barrel? A letter?)

Avatars of people are currently simplified figures (discussed in the *Abstraction* section of *Design Choices* above). Each pose is a keyframe in a Flash MovieClip, that is instantiated, placed, and scaled within the bounds of the frame based upon the XML panel description. This approach appears to be the same as ATR Labs' approach in ComicDiary; the future work section of their paper suggests making articulated figures that can be posed instead of instantial assets. I would agree: should more detailed characters be used, an articulated figure would be the preferred method of implementation. However for simpler characters and rapid prototyping purposes, the initial effort of building a skeleton and posing and tweaking it can be far more time-consuming than simply drawing the stick figures.

Because current data analysis on digital images and audio does not allow for reliable recognition of the identities and exact relative locations of other people (particularly others who are not wearing computers), the algorithms for relational positioning of characters and the word balloons tied to them given in the Comic Chat paper (Kurlander, et al. 1996) are not as applicable to this work. This prototype currently avoids using *emanata* altogether and depends on the body language of the characters. In the words of Will Eisner (1985), "The use of expressive anatomy in the absence of words is less demanding because the latitude for the art is wider. Where the words have a depth of meaning and nuance the task is more difficult." Adding dialogue and similar symbology to future implementations will doubtlessly be informed by the Comic Chat research.

In prior works, the background has been kept as a predrawn instantial asset—a clip art piece to match the location of the characters—that rarely changed, regardless of

the movement of the characters in the panels relative to the background. Ideally, in this software, this background should be tied to a geometric representation of the map, so that squarish blocks signifying buildings at least correspond to the scale and position they should be in judging from the position and orientation of the subject character on the map. Backgrounds are quite important to orientation in the world behind the frame of the comics panel, so this is an issue which should be quickly addressed.

Backgrounds should also reflect the climatalogical conditions of the events. There are sites which have XML (or otherwise extractable) data on historical weather (such as http://www.washingtonpost.com/wp-srv/weather/historical/historical.htm or http://www.noaa.gov/pastweather.html) although it is probably best if this information is added into the activity or locative XML data. At the very least, the color of the sky should be changed to reflect the time of day.

Rules for External Panel Layout

External panel layout describes a panel's shape, its size (amplitude), and the size of the gutters around it. As mentioned earlier, leveraging external panel layout, by varying panel sizes as comics do, can qualitatively improve summarizations (Boreczky 2000). The following explorations of shape, size, and spacing are shown in further detail in Appendix 2.

While the heuristics below can be helpful, the fact that most comics creators today use only standard size panels, evenly distributed, with standard rectangular borders is something to heed. Panel variations are best used in moderation and relative to the other panels in the sequence. If the computer produces a comic in which there are back-to-back-to-back giant jagged-edged panels (the rare panel shape described by Eisner in Fig. 10) with wide gutters, then the external panel layout loses its usual significance, and a small, rectangular panel becomes the unusual event. Therefore, since external panel layout is *relative* to each run or collection of runs, it may (and should) change when the comics summarization is scaled from a day worth of data to a month or a year.

Working in external panel layout, one notices that there are many ways to accomplish the same effect. A large rectangular panel expressing significant action could also be expressed as a smaller panel with a jagged-edged border to express relatively similar narrative significance. In a more appropriate example, wide panels, conveying a sense of timelessness, could be used interchangeably with thin open panels that have no border, should horizontal space on the screen be an issue. Thus, there appear to be several common narrative devices that external panel layout supports, regardless of whether varying the size, shape, and spacing of panels:

- Temporal jumps (*Flashes*)
- Simultaneous actions (*Meanwhiles*)
- Long duration actions (*Grinds*)
- Temporal slowdowns (*Loiters)*
- Sudden or emotionally intense actions (*Bursts*)
- Establishing shots or emphasis of scale (*Forests*)
- Important details (*Twigs*)

These narrative devices, the panel shapes, sizes, and spacings that support them, and the priorities with which each variation should be used to accomplish said narrative devices, are specified in Appendix 3.

Border Shape

While rectangular frames are the most common type of panel border in comics, the medium can employ a wide variety of shapes to frame pictures and aid storytelling.

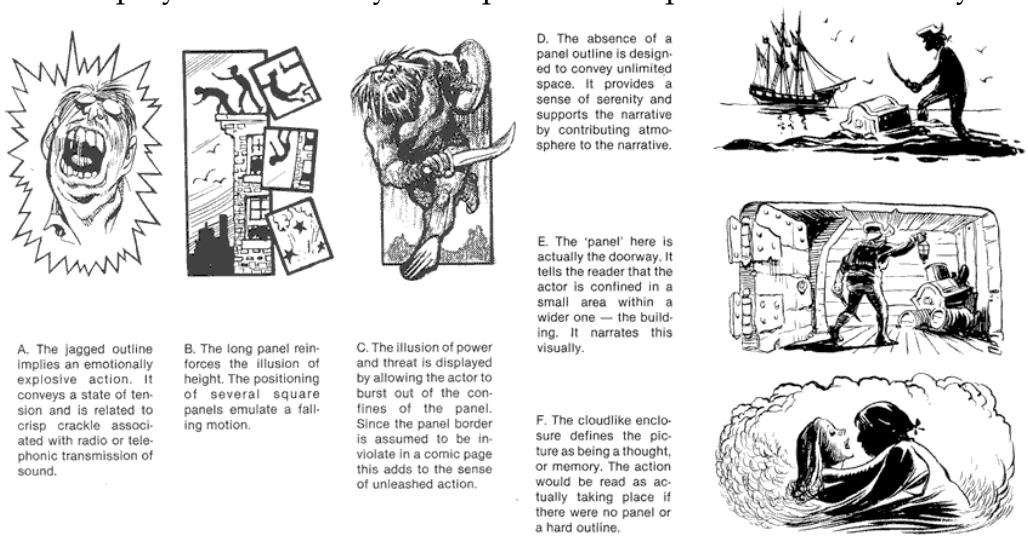


Fig. 16: Examples of panel border as a narrative device, from pages 46-47 of *Comics & Sequential Art* (Eisner 1985)

Conventions in panel border shape have come to be relatively standard over a hundred or so years of use, although their usage is dependent upon the artist. If a bookstore customer were to flip through the pages of all comics currently on the shelves today, she would find that the majority of artists use the rectangular border nearly exclusively, and while each artist may use one or more different border styles, it is extremely rare for one artist to use ALL of them. (The notable exception being the body of work of Will Eisner, who experimented with the frame of the panel as a primary means of expression in nearly all his comics.)

Appendix 2 outlines these Panel Variations. The most important thing to note from these variations is that the shape of a panel can be very useful in differentiating the events contained inside one particular panel from the events contained in the rest of the normal, rectangular-box narrative. This is the one way in which comics creators do actually use this convention consistently; even though one artist may use a `roundrect` for a thought balloon and another artist may use `roundrect` for all panels presented from a separate POV (point of view), neither method is wrong.

In addition to these frame shapes, panels can be stretched or squashed to show a longer or shorter duration in time or to accentuate horizontal or vertical motion. These rules will be discussed in the next section (on panel size).

Panel Size

Panels are usually scaled for one of three reasons:

1. the relative semantic importance of the panel contents (e.g., significant actions or emotional impact)
2. the amount of detail to present at that moment or location in space is far greater or far less than necessary
3. the scene setting has changed, and a larger panel is necessary to show detail in much the same way as the use of an establishing shot in film

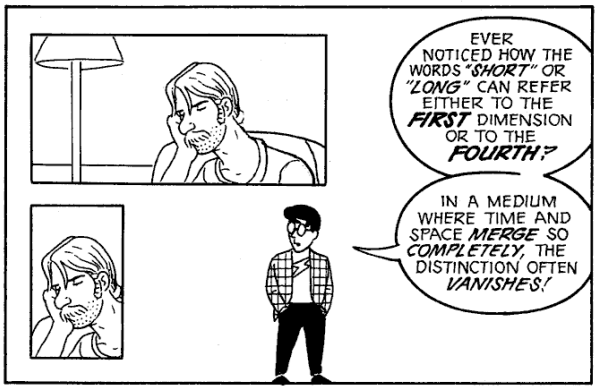


Fig. 17: McCloud ponders time in panel widths (1993, p. 102)

Panels may also be sized to fill out space on a printed page (or screen), but this should be done with care so as to not disrupt the purposeful sizing of panels. In this implementation, size should be determined in much the same way as Video Manga, by ranking the activity by rarity and duration (Uchihashi & Foote 1999). Less common, rarer activities will warrant larger panels, for (particularly in the case of military troops on patrol) it is the unusual events that bring narrative to the monotony of daily activities. Longer duration activities will also receive a higher importance ranking, so that panels are smaller (or not generated at all) for extremely common activities of extremely short duration. Unlike Video Manga, our panels are generated after all ranking occurs, meaning that instead of serving to choose frames to omit, our process chooses which frames to create.

Panel width is most often varied to reflect temporal density. Thinner panels generally represent shorter segments of time, much like jump-cuts in film. Wider panels represent longer durations of time (or in a few cases, or draw out a moment for emotional impact as if it is frozen). When dialogue is involved, longer (and taller) panels can be a way used as a way to juggle lots of text, but this practice is a holdover from the noninteractive printed page; text can now be placed in rollover captions that economize the amount of text squeezed onto the viewable comic. In this implementation, wider panels should by and large be granted to activities of longer uninterrupted duration.

Panel height is varied mostly to reflect vertical size or motion. As shown in the Eisner example, tall panels can be used to show falling from a height, but they can also be used to show a character looking up at rooftops, running up or down stairs (to emphasize the vertical motion), or standing next to a tall statue. In this implementation, panel height should be increased for "going up or down stairs" activities.

Gutter Size

Gutter size is varied to influence the reading rhythm of the comics:

- to emphasize the direction of the panel flow and the read order of panels
- to step out of the flow of time in the panels, either by pausing, skipping ahead, or slowing down time, most often with the intent …

- to emphasize the importance, emotional impact, silence, openness, or isolation of a panel

Panel frequency can be used to show the relative temporal positions of data collections (a temporal map of activity runs), as well as to highlight lacunas (gaps in collection) and precede extremely long duration or extremely rare panels for emphasis. Varying the gutter size to cluster sets of panels together is especially important in long-scroll digital comics which are not placed together by nature of the dimensions of the printed page.

Panel frequency in this prototype should be kept to its simplest use. Gutters are already calculated by default to be twice the small margin from the edge of the grid squares that the panel leaves itself. Other distances between panels are usually specified in units of that same grid. Examples of these conditions can be found in Appendix 2.

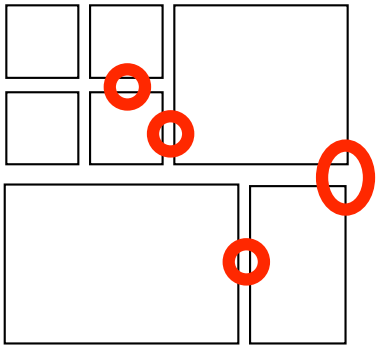


Fig. 18: Using the gutter sizes to emphasize the read flow is something Video Manga does rather subtlely

User Interface

The proposed user interface is depicted in Appendix 8. While most of the screen real estate is given to the comics panels themselves, there are also stubs for a timeline at the top of the screen, expanding panels to show a map, as well as captions and comments, and links to media assets associated with the time period depicted by a panel.

The ability for the wearable user to add annotations for panels aligns with Schank's *commentary* aspect of *elaboration* in the creation of stories and memories (Schank 1990). The ability to put in one's own thoughts to the comics narrative that the computer creates will further assist this tool in being a memory aid for the user.

Design Issues and Future Work

One of the biggest design issues of this project was properly scoping the problem. The initial project idea was so nebulous and overambitious, that the goals for the project were repeatedly refined (or scoped down as time became shorter and shorter). In the end, the final project offers only variations in external panel layout to contribute to the efforts of ComicDiary and Video Manga.

While I created loose formats for XML specifications to be used with this tool, these specifications should probably be formalized at least into XML Schema. Taking such action will help to prevent the possibility of namespace conflicts down the road, and will also force the specification to be made tighter. Currently, elements have more subelements than are likely necessary (e.g., *action* has an *activity*, in the anticipation that multiple subject-activity-object depictions would occur in a single panel), and the design of the XML specification could use further review and revision.

In my initial proposal, I had planned to complete this project in time to run some preliminary evaluations using the prototype. These evaluations, aside from formative evaluations throughout the process, have not yet been conducted. While the CCG has talked to users about which activities they think are most important to be able to identify and highlight in summaries, I have accomplished little user-research (aside from personal anecdotal evidence) on how useful this product would be to soldiers and intelligence officers.

One concern that I do have, however, is that this software tool may be used independently by higher ranking officers to try to piece together their own account of events. The software, in my opinion is intended to be a memory aid, so without at least passing through the filter of the user's own annotations, using such software to make actionable decisions is dangerous. This tool should make it apparent at all times which story skeleton it is using to create its narrative, so that the end user is well aware of the biases that may be introduced by following a particular mission type or POV.

I would also caution against the use of this project by ranking officers who want to use this summarization tool to conduct their own personal intelligence analysis on the experiences of soldiers in the field. While these officers are undoubtedly well-educated, experienced, and wise, they generally come from career fields that receive no training in intelligence analysis, and are often not prepared to take into consideration the ambiguities and biases that plague the discipline (Heuer 1999). This tool may help provide trained intelligence analysts with another perspective or another modeling tool of "ground truth," but making any decisions based solely or primarily on the summaries generated by this software is a bad idea.

Future work in this field could include not only more work in improving the narrative fidelity of the comics summaries generated (perhaps through the stochastic approach recommended by Dr. Starner), but also in adding more comics-like elements.

- In Chapter 5 of *Understanding Comics*, McCloud discusses the use of emotional qualities of lines, such as raw and jagged brush strokes to depict anger, soft cross-hatches to depict serenity, and so on. Procedural depictions of emotional state drawn from collected sensor data could be used as backgrounds of panels (or their border line styles) to further highlight salient panels in summaries, whether on the page or the screen. There have already been several photo manipulation papers on this topic and one paper (Hwang 2006) that proposes applying such line effects to a video stream specifically to create comics.

- Symbology and *emanata*, such as word balloons and depictions of other sensory (auditory, olfactory, etc.) information, could be added to panels once subjects of panels can be identified with more certainty. Granted, when panels are generated from scratch as opposed to using pre-existing images, it is much easier to find the location of the depiction of particular people in the comics panel. Much of this symbology is already specified in ComicsML (McIntosh 2003). Furthermore, there is research being done as a part of the ASSIST initiative to

identify voice energy and language to identify shouting, excited talking, and which language speech is in, and even to note nonspeech high-energy sounds such as gunshots or explosions. This identification information could be combined with small speech bubbles positioned near the heads of characters in the panel. These bubbles need not have roots (the triangular stem that points to the speaker) if the speaker cannot be identified, such as in the case of a murmuring crowd. Examples of possible word balloon variations are shown in the figure below.

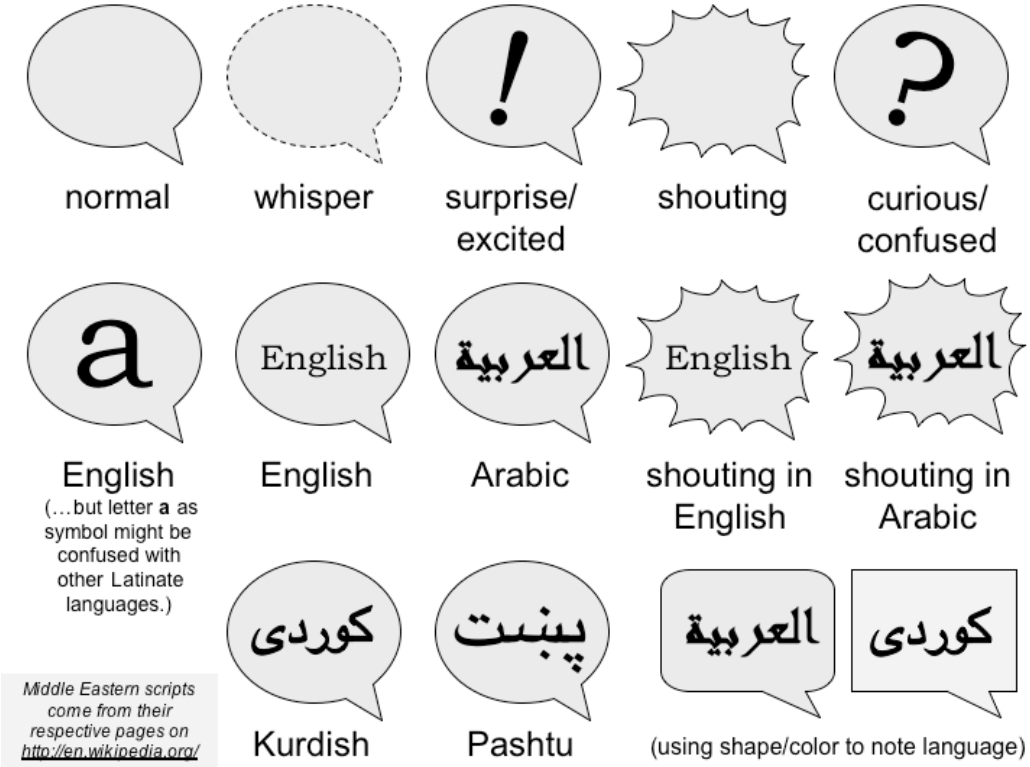


Fig. 19: Possible variations of speech balloons that could be used to express speech analysis data.

Sound analysis could also be used to determine context for the background of panels: if there is the sound of a cacophony of yelling protesters, a crowd could be drawn into that panel. (This approach could bring up rendering issues, however, because you wouldn't want the crowd disappearing from one panel to the next, should a momentary hush fall on the crowd.)

Further research could also be conducted to discover the universality of certain symbology. In my proposal, I noted the work of Japanese *manga* theorist Fusanosuke Natsume, who has written about the importance of layout in telling stories (Kite 2003) and has written about many other stylistic trends in *manga* creation, but his critical writings are largely untranslated into English. With the particular popularity of Japanese comics in North America now, it may not be long until comics on this continent also become fluent in Japanese *manga* symbology.

- Characters should probably be posable puppets, with skeletons and articulated joints, instead of static keyframes. (This suggestion is also made in the Future Work section of the ComicDiary paper (Sumi, et al. 2002).) Such model manipulation would allow a greater degree of procedural expression, if only to make the panels look slightly more natural and less like clip-art. By this suggestion, I do not mean that the characters should be posed to match the raw accelerometer data. By creating models of characters, whether 2D or 3D, that can be manipulated and posed by methods, there is a steeper curve in developing the initial artwork, but groundwork is laid for any future activities that analysis is able to parse from the wearer's experiences. These future activities will be able to be enacted in panels with far less effort, simply by sending a list of posing data to the puppet model, rather than requiring an artist to draw another keyframe. Although simplicity of avatars' appearances should also be maintained, this implementation would make it easier to "skin" characters with different appearances.

- External layout is currently highly dependent upon the renderer rather than on the description. This decision was made to simplify the comics description XML, but it could lead to limitations in layout further down the line as the code to generate the comic becomes more expressive. Future work could explore allowing the narrative generation component more specific control over where it places panels, aside from simply varying size and spacing. Being able to overlap and layer panels or place panels at certain coordinates on the grid could possibly introduce new narrative effects. For example, making the screen itself a panel, by placing a large establishing shot as a background for smaller panels on the screen could help to provide a sense of place without necessitating a large panel explicitly in the midst of the narrative flow. Gutters would also no longer have to be horizontal or vertical, but could be diagonal, as in *Fig. 5*, creating more fluid and organic panel read flows. All of this could be done while still maintaining the 10x6 grid, keeping some constraints to maintain a structure for the screen "page".

- While the current specification outlines a way for the wearable user to flesh out the comics narrative after the fact, through annotations, perhaps the user can tweak the narrative by dialing up or down the importance (and therefore size) of any particular panels. The computer would then have to reevaluate the relative importance of all of the panels, and recalculate the narrative. This may be too much tweaking power for the average user, but it is an option that might offer more agency to the user in the creation of their summary, combining an aspect of human-authored-comics tools, while still leaving narrative creation in the hands of the computer.

Finally, further work could be done to completely flesh out this pseudocode and rough prototype into a fully functional software interface.

While comics have a bit of a stigma associated with them—mostly because of the kinds of stories predominantly told with the medium in the US—their underlying principles are entrenched in centuries of storytelling and design experience, and their particular presentation is ingrained in our cultural consciousness. As a medium for conveying information quickly, their affordances are largely untapped in summarization software, save the few tools covered in this paper. I think that further research in this area will only continue to benefit user communities with far too much information and far too little time to sift through it, whether they be academic, military, or just regular people with abundant digital records of their lives.

References

Barber, J. (2002). *The Phenomenon of Multiple Dialectics in Comics Layout*. Retrieved October 16, 2005, from http://www.johnbarbercomics.com/write_layoutintro.htm

Boreczky, J. et al. (2000). *An interactive comic book presentation for exploring video*. Retrieved September 24, 2005, from http://www.fxpal.com/publications/FXPAL-PR-00-086.pdf

Cohn, N. (2003). *Early Writings on Visual Language*. Carlsbad, CA: Emaki Productions.

Cohn, N. (2003). *A Language by Any Other Name....* Retrieved October 21, 2005, from http://www.emaki.net/essays/ClarifyingVL.pdf

Cohn, N. (2003). *Visual Syntactic Structures, Part 1*. Retrieved October 21, 2005, from http://www.emaki.net/essays/VSS1.pdf

Cohn, N. (2005). *A Force of Change*. Retrieved October 21, 2005, from http://www.emaki.net/essays/forceofchange.pdf

Cohn, N. (2005). *Initial Refiner Projection*. Retrieved October 21, 2005, from http://www.emaki.net/essays/Refiners.pdf

Cohn, N. (2005). Reframing Comics. *Comixpedia*. Retrieved October 21, 2005, from http://www.comixpedia.com/modules.php?op=modload&name=News&file=article&sid=2640

Cohn, N. (2005). *A Visual Lexicon*. Retrieved October 21, 2005, from http://www.emaki.net/essays/visuallexicon.pdf

Dotinga, R. (2005). Holy Homework! Comics Hit schools! *Wired News*. Retrieved October 31, 2005, from http://www.wired.com/news/culture/0,1284,68244,00.html

Eisenstein, S. (1949) *Film Form*. Translated by Jay Leyda. Orlando: Harcourt Brace & Company. (Referenced in quote from Barber paper; citation here as courtesy.)

Eisner, W. (1985). *Comics and Sequential Art*. Tamarac, FL: Poorhouse Press.

Eisner, W. (1996). *Graphic Storytelling*. Tamarac, FL: Poorhouse Press.

Electronic Arts. (2001). *The Sims Comic Strip*. Retrieved October 16, 2005, from http://thesims.ea.com/us/about/expansionpack/spotlight/

Gelernter, D. (2000). *The Second Coming – A Manifesto*. Retrieved September 25, 2005, from http://www.edge.org/3rd_culture/gelernter/gelernter_p1.html

Hadley, M. (2001). *Cage*. Retrieved October 16, 2005, from http://www.geocities.com/Petsburgh/Haven/5337/

Heuer, R. J., Jr. (1999). *Psychology of Intelligence Analysis*. Washington, DC: Center for the Study of Intelligence. Retrieved April 13, 2006, from http://www.cia.gov/csi/books/19104/

Horn, R. (1998). *Visual Language: Global Communication for the 21st Century*. Bainbridge Island, WA: MacroVU, Inc.

Horrocks, D. (2003). *The Perfect Planet: Comics, Games, & World-Building*. Retrieved September 14, 2005, from http://www.hicksville.co.nz/PerfectPlanet.htm

Hwang, W., et al. (2006). *Cinema Comics: Comics Generation From Video Stream.* Retrieved May 3, 2006, from http://pearl.cs.pusan.ac.kr/publication/HwangWI2006GRAPH.pdf

Intel Corporation. (2005). *Intel Digital Lifestyle Report*. Retrieved October 30, 2005, from http://www.intel.com/cd/corporate/pressroom/emea/eng/233325.htm

Kay, A. (1995). *Powerful Ideas Need Love Too! Remarks to a Joint Hearing of the Science Committee and the Economic and Educational and Opportunities Committee*. Retrieved October 13, 2005, from http://lcs.www.media.mit.edu/groups/el/events/love-too.html

Kite, H. (2003). *Art on the fast track*. Japan Times, 23 March 2003. Retrieved October 26, 2005, from http://www.japantimes.co.jp/cgi-bin/getarticle.pl5?fl20030323a3.htm

Krashen, S. (1996). *Comic Book Reading, Reading Enjoyment, and Pleasure Reading Among Middle Class and Chapter I Middle School Students*. Retrieved September 28, 2005, from http://www.sdkrashen.com/articles/comicbook/menu.html

Kurlander, D., Skelly, T., & Salesin, D. (1996). *Comic Chat*. Retrieved October 24, 2005, from http://smg.www.media.mit.edu/classes/VirtualSociety99/temp/p225-kurlander.pdf

Lewis, J. (2003). Memory Overload: As hard drives get bigger and cheaper, we're storing way too much. *Wired*, 11.02. Retrieved October 25, 2005, from http://www.wired.com/wired/archive/11.02/view.html?pg=2

Lim, C., et al. (2001). *Digital Propp. Final project for Professor Lewis Seifert's Fairy Tales and Culture course at Brown University*. Retrieved October 31, 2005, from http://www.brown.edu/Courses/FR0133/Fairytale_Generator/

Mann, S. (1998). *What is a Wearable Computer?* Excerpt from "Wearable Computing as means for Personal Empowerment," keynote address at 1998 International Conference on Wearable Computing, May 1998. Retrieved October 30, 2005, from http://about.eyetap.org/fundamentals/

Mathes, A. (2004). *Folksonomies – Cooperative Classification and Communication Through Shared Metadata*. Retrieved October 26, 2005, from http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

McCloud, S. (1993). *Understanding Comics.* New York, NY: Kitchen Sink Press/HarperPerennial.

McCloud, S. (2000). *Reinventing Comics*. New York, NY: Paradox Press.

McCloud, S. (2003). DVD of talk given to Georgia Institute of Technology, recorded by Danny Muller, October 10, 2003.

McCloud, S. (2005). Telephone conversation, September 23, 2005.

McIntosh, J. (2003). *ComicsML: A proposed simple markup language for online comics*. Retrieved September 26, 2005, from http://www.jmac.org/projects/comics_ml/about.html

Memmott, M. (2005). 'Milbloggers' are typing their place in history. *USA Today*. Retrieved October 26, 2005, from http://www.usatoday.com/news/world/iraq/2005-05-11-milblogs-main_x.htm

Neurath, O. (1946). From Hieroglyphs to Isotypes. Intro by Rotha, P. *Future Books No. III*. Retrieved September 29, 2005, from http://www.fulltable.com/iso/is03.htm

PlanetWide Games. (2005). *Comic Book Creator*. Retrieved October 26, 2005, from http://planetwidegames.com/products/comicbookcreator/

Pound, J. (2004). *About Random Computer-Generated Comics*. Retrieved October 26, 2005, from http://www.poundart.com/art/randcomix/about.html

Schactman, N. (2004). Pentagon Revives Memory Project. *Wired News*. Retrieved October 25, 2005, from http://www.wired.com/news/print/0,1294,64911,00.html

Schank, R. (1990). *Tell Me A Story*. New York, NY: Charles Scribner.

Shiga, J. (2003). *Every Dog Has His Day* description. Retrieved October 27, 2005, from http://www.shigabooks.com/interactive/every.html

Smith, G. (2004). Folksonomy: Social classification. *Atomiq*. Retrieved October 26, 2005, from http://atomiq.org/archives/2004/08/folksonomy_social_classification.html

Stern, A., et al. (2004). Comic Interaction. *GrandTextAuto*. Retrieved June 6, 2005, from http://grandtextauto.gatech.edu/2004/09/02/comic-interaction/

Sumi, Y., et al. (2002). ComicDiary: Representing Individual Experiences in a Comics Style. *Proceedings of the 4th International Ubiquitous Computing Conference, UbiComp 2002*, p. 16. Retrieved October 25, 2005, from http://www2.mis.atr.jp/~sumi/papers/ubicomp02.pdf

Tufte, E. (1990). *Envisioning Information*. Cheshire, CT: Graphics Press.

Turner, J. (2002). *Bright Blue Morning*. Retrieved April 11, 2006, from http://www.jasonturnerproject.com/blue/brightblue.html

Uchihashi, S., & Foote, J. (1999). Summarizing Video Using a Shot Importance Measure and a Frame-Packing Algorithm. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (Phoenix, AZ)*, Vol. 6, pp. 3041-3044. Retrieved April 8, 2006, from http://www.fxpal.com/publications/FXPAL-PR-99-134.pdf

Uchihashi, S., et al. (1999). *Video Manga: Generating Semantically Meaningful Video Summaries*. Retrieved October 25, 2005, from http://www.fxpal.com/publications/FXPAL-PR-99-136.pdf

Verbarg, K. (2006). Tell Me A Story. *chisquare_kaisquare*. Retrieved April 26, 2006, from http://chisquarekaisquare.blogspot.com/2006/01/tell-me-story_25.html

Walker, M. (1980). *The Lexicon of Comicana*. Port Chester, NY: Museum of Cartoon Art. Derivative information retrieved September 28, 2005, from http://en.wikipedia.org/wiki/Comics_vocabulary

Wikipedia.org. (2004). Vladimir Propp. Retrieved October 31, 2005, from http://en.wikipedia.org/wiki/Vladimir_Propp

Wildbur, P., & Burke, M. (1998). *Information Graphics: Innovative Solutions in Contemporary Design*. London, UK: Thames & Hudson.

Williams, R., Barry, B., & Singh, P. (2005). *ComicKit: Acquiring Story Scripts Using Common Sense Feedback*. Retrieved September 14, 2005, from http://web.media.mit.edu/~push/WilliamsBarrySinghIUI2005.pdf

Further references at http://idt.gatech.edu/~jalderman/slog/?p=2 and http://del.icio.us/jalderman/thesis.

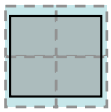
Appendix 1: Summary of Comics Generation Tools

Name	*Type*	*Description*	*Features*	*Citation*
Video Manga http://xrl.us/vidmangapaper1	1	Summarizes video clip in comics form, as a way for users to more quickly browse a long video clip for specific content.	Different panel sizes and layout, determined semantically from length of subclips. Panel shapes same; uses popup captions; lacks elements such as word balloons; still leads peers in external panel layout among Type 1 tools.	Uchihashi, Boreczky, et al., FX Palo Alto Lab
ComicDiary http://xrl.us/comicdiary http://xrl.us/comicdiarypaper	1	Creates a humorous anecdote about a sensor-rigged conference-goer's experiences, in comics form.	Summarizes experiences in a way that entertains and encourages communication between users. Draws from several sources of sensor data (PDA notes, surveys, electronic business card exchanges, IR sensors) to generate story.	Sumi, et al., ATR Japan
ComicKit http://xrl.us/comickit	1	Queries AI common sense database to help user create comic.	Determines what other contents belong in a panel, based on the contents placed there by a user. Designed more as a test of AI than a test of comics generative abilities of computer.	Williams, et al., MIT Media Lab
Squares http://slackworks.com/~cog/	1&2	Generates new stories based on which props are dragged into six panels.	Panels and layout do not change; new stories are generated within the same framework and basic setting.	Gingold, Georgia Tech (Maxis)
Comic Chat http://xrl.us/mschatpaper	1&2	Creates long-scroll comic strip transcript of multi-user Internet Relay Chat (IRC) conversations.	Changes posture and facial expression of users' avatars in chatroom; places word balloons in natural reading order; automatically zooms "camera" to frame the people talking; automatically parses text into panels procedurally. Panels all same size.	Kurlander, et al., Microsoft
Comic Book Dollhouse http://slackworks.com/~cog/	(1&)3	Creates a comic story from the props used with the poseable characters placed in the panels.	Lets users choose two characters, and a prop that has associated verbs, to write out a story in comics panels. Also allows for posing characters (like dolls).	Gingold, Georgia Tech (Maxis)

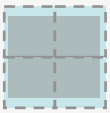
The Sims Family Albums http://xrl.us/simscomics	1&2	Original game let players take photos of Sim families and post online. Users added narratives and made comics.	Sims characters are simulated within game, but player may snap pictures of them, and write narrative of what is happening. Very loosely a comic; all panels are subtitled with text and have same size.	Wright, Maxis/EA
Comic Book Creator http://xrl.us/cbcreator	1&3	Combines screen capture utility with library of comics symbols to let users make comics from the games they play.	Compatible with nearly every PC game. Lets users make comics from their favorite role-playing or massively multiplayer games.	PlanetWide Games
ComicLife http://plasq.com/comiclife	3	Application for loading images into comics panels and adding word balloons, sound-effects, etc..	More creative freedom than most other applications on this list for panel size, shape, and layout. Lets users crop and zoom photos/ drawings to fit panels. Print or export to share on Web.	Pearson, Grant, et al., Plasq.com
MangaStudio http://xrl.us/mangastudio http://xrl.us/comicstudio	3	Application for drawing, laying out, lettering, and stylizing comics.	Appears to be the Western import of Japanese ComicStudio, an extremely popular comics-creation toolset, akin to comics-focused Photoshop or Illustrator.	eFrontier (and TOPPAN and CelSys)
Balloonist http://xrl.us/balloonist	3	Application for laying out and lettering comics, geared toward writers.	Allows creation of a comics page from template or by design; automatically wraps and buffers text inside word balloons; will import ComicsML; allows gutters between panels to be adjusted.	Smith & Tinker's Technology
Ran Dum Comics http://xrl.us/randum	2	Proprietary code that procedurally generates comics dialogue, panels, and characters.	Generates comics that can sometimes make sense. Pound codes programs to create his random comics.	Pound, PoundArt.com
Online Comics Generators	2&3	Web sites that use Flash or server-side scripts to generate comics from limited user input.	Can range from tools similar to ComicLife, to tools similar to Ran-Dum. Lets non-artists create comics, but functionality is severely limited. Panels are generally a uniform size, and inhabitants of panels are usually clip-art in nature.	(Various. Please see idt.gatech.edu/~jalderman/slog/ and del.icio.us/jalderman/comix+thesis for examples.)

Appendix 2: Panel Variations

Panel Shapes

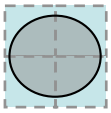


Title: ***Rectangular***
Nickname: `rect`, `normal`
Description: Normal rectangular border with line border
Used when? This is the standard panel type used in most comics.
Specific examples: This should be the default panel border type in comics generated, unless other conditions specifically apply.



Title: ***Open*** or ***Full-bleed***
Nickname: `open`, `bleed`
Description: Rectangular panel with no line border
Used when? Often used without a background, and usually between two other rectangular panels, this type of panel focuses attention on the subject of the panel (whether person(s) or object). Will Eisner claimed that this style of panel expresses an expanse of unlimited space, implying serenity and atmosphere to the narrative (1985). Scott McCloud notes that the lack of a border, regardless of size, gives an impression of timelessness (1993). In both cases, it is a step outside the flow of time and/or space as depicted in the comic. Could be used for long durations, especially when horizontal space is at a minimum.
Specific examples:

- Showing actions of another wearable user or another object or person in the narrative that are simultaneous to the current action
- Showing a long duration event, such as the user baking in the sun standing on duty at a monotonous checkpoint



Title: ***Circlular***
Nickname: `circle`
Description: Circular or ellipsoid panel, usually small or medium-sized, with line border
Used when? Used to highlight single item, usually a close-up on a person's face or hand, a marking on their clothes, or another detail. Common examples are the highlighting of a clue in a mystery, a view through a peep hole, the head of a person when they suddenly appear, or the head of a person when they make a notable facial expression. If more than one aspect in a row need be highlighted, it is more common to use multiple rectangular panels; circular panels are generally not used back to back. (One

reason for highlighting might be foreshadowing.)
Specific examples:

- If it is important to the narrative to show a suspect's tattoo, a footprint, a face, or some other kind of element that is either the narrative "payoff" of some earlier thread or an element of foreshadowing, a circle panel could be used.
- A close-up of the timer of a bomb hidden in roadside debris
- A close-up of a hand touching a light-switch
- A close-up of a boot tripping a tripwire

Title: ***Rounded-rectangular***
Nickname: `roundrect`
Description: Rectangular panel with rounded corners and a line border
Used when? While some artists use this as their default panel type, it is most commonly a convention for a flashback or dream sequence (a more manageable, simpler-to-draw version of the thought balloon frame shown in Fig. 10). Could also be used to show a different POV or simultaneous action.
Specific examples:

- A user shakes hands with the mayor of a town and recalls the last time that he met the mayor three weeks ago (rendered in a roundrect panel), noting the mayor's change in body language from hostile then, to jovial and friendly now.
- In the middle of a user's comics narrative, while they are standing checkpoint duty, a panel is shown from the perspective of another soldier who is on a sidestreet not far from the checkpoint, chasing a suspect toward the checkpoint. (This provides foreshadowing as well as combining the wearable data of two users.)

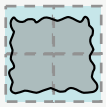
Title: ***Jagged-edged***
Nickname: `blast`
Description: Roughly rectangular or ellipsoid panel, but with triangular spikes zig-zagging its edges (as shown). Spikes' lengths are never all the same, but are often randomly varied, staying within a range of just-past-the-gutter to an arbitrary internal margin.
Used when? Used to describe events of high emotional impact such as fright, danger, surprise, shouting, or a sudden burst of movement. May also be used to emphasize uses of actual high-

energy items, such as bright lights, high-voltage wires, machine guns, etc., but usually emphasizes these items only when they have a high emotional impact as listed above.

Specific examples:

- A presence patrol spots an insurgent they are seeking
- A cordon and search busts down the door of a house in a raid
- Two soldiers are standing and talking when suddenly there is gunfire directed at them
- An explosion occurs (panel shows explosion)

(Others, not currently implemented)



Title: ***Wavy***
Nickname: `wavy`
Description: Roughly rectangular panel, but shaky lines that oscillate sinusoidally a few pixels in and out as shown.
Used when? Can be used to express emotional states of confusion, fear, fever, or dread, or physical shakiness or heat.
Specific examples:

- An explosion has just occurred and damage assessment is being made
- A soldier has been hiking for hours and hours in the hot sun and is exhausted and weakened
- Soldiers are nervous because the streets are empty and they think an ambush is about to occur

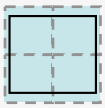
Title: ***Rotated***
Nickname: `rotated`
Description: Rectangular panel rotated left or right between 0˚ and 30˚, and scaled to fit within the boundaries of the square grid.
Used when? Can be used to express fast action, particularly spinning/turning, or dizziness. Could be used to subtlely show abrupt changes in direction measured with digital compass.
Specific examples:

- A suspect has been sighted by a fireteam, and a rotated panel shows another fireteam turning to assist pursuit
- A soldier has been thrown across a room or into a wall by an explosion (similar to the fall from height in the Eisner example, Figure 10).

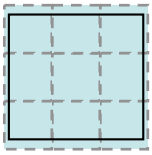
Panel Sizes

Unless specified otherwise, default gutters on all panel size categories below: Panel is inset 5% (3 pixels on a 60x60-pixel grid square) on any sides touching other panels in the reading row, and 8% (roughly 5 pixels on a 60x60-pixel grid square) on sides touching the edges of a reading row.

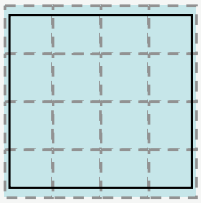
Name: `small`
Dimensions: 1 x 1 (1 grid square)
When used? Shows less significant or repetitive activities, or small aspects of a larger panel.



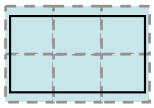
Name: `normal`
Dimensions: 2 x 2 (4 grid squares)
When used? Default panel dimensions …should be panel used most often.



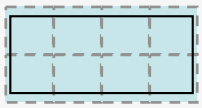
Name: `large`
Dimensions: 3 x 3 (9 grid squares)
When used? Panel is more significant relative to other panels in narrative (see *External Panel Layout* section in main document).



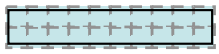
Name: `xlarge`
Dimensions: 4 x 4 (16 grid squares, or larger, up to 6x6—the largest that will fit on screen at one time—, as long as both width and height are the same)
When used? Should be very rare …when lots of unusual or significant events are used in a narrative and the panel size across the board has not been adjusted down, this panel size expresses greater narrative significance. When panel-packing algorithms (Uchihashi & Foote 1999) are in effect, this panel may also be a result of relative size adjustments to justify panels to margins.



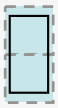
Name: `wide`
Dimensions: 2 x 3 or 3 x {4,5} (6 or 12-15 grid squares)
When used? Panel has onger duration than usual or emphasized horizontal motion. Height determined by space available, or relative narrative significance with other panels, but height should NOT be greater than or equal to width.



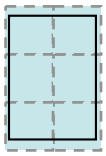
Name: `xwide`
Dimensions: 2 x {4,5,6} or 3 x {6,7,8} (8-12 or 18-24 grid squares)
When used? Used for long duration, little-change-in-action events or to accentuate horizontal motion. Width may change due to space available; height should change due to space available or relative narrative significance, but height should always be at maximum half of width.



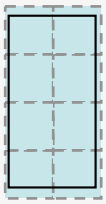
Name: `fullwide`
Dimensions: {2,3,4} x 10 (20-40 grid squares, full width of 10x6 screen)
When used? Used for extremely long duration, such as an extremely long period of monotony, or for wide-angle vistas, such as a pan around a room or a panorama of a new location, usually at the beginning of a scene or upon arrival at that location. Height determined by space available or by relative narrative significance ranking.



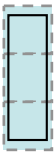
Name: `thin`
Dimensions: 1 x 2 (2 grid squares) [Since we are working with whole squares, there is no thin panel width for small panels that are 1x1 grid square. Panels of `tall` dimensions may be used for `thin` if the relative context of panel sizes is 3 high or larger, although this should be extremely rare.]
When used? Used for temporal jump cuts; brief reactions; or relatively very short-duration (but still important) elements within the narrative.



Name: `tall`
Dimensions: 2 x 3 or 3 x {4,5} (6 or 12-15 grid squares)
When used? Used to emphasize vertical motion (e.g., stairclimbing) and relative height (e.g., full-body shot of suspect to show height next to doorframe). May be stretched to fit space available, but height should always be greater than width, but less than twice width.



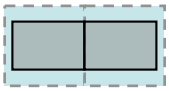
Name: `xtall`
Dimensions: 2 x {4,5,6} or 3 x 6 (8-12 or 18 grid squares)
When used? Same as `tall`, but for greater motion or height comparison. May be stretched to fit space available, but height should always be greater than or equal to width.



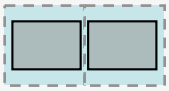
Name: `tallthin`
Dimensions: 1 x {3,4} (3-4 grid squares)
When used? Very rare; used for extremely short duration moments of emphasis of height or vertical motion, e.g., a split-second glimpse at something in the sky, like a helicopter established in a previous tall panel, just to show that it is still there. Particularly used when other panels are `normal` sized; however, very likely **not** to be used at all.

Panel Spacing (Gutter Sizes)

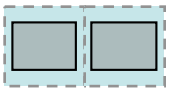
Gutters sizes are applied to one side of the panel: Gutters can be above or left or below or right, but are usually identified as "`gutter-before`" and "`gutter-after`"…the computer should determine where the previous panel is, in relation to the current panel, and make adjustments accordingly. It is fully possible for a panel to have an `xwide` gutter before it and a close gutter after it. Also, for small gutter sizes, they work best if the same gutter type precedes the following panel.



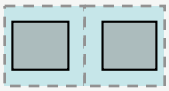
Name: `none`
Dimensions: No space, no inset, 0 grid squares
When used? Used for stylistic purposes, or to give the sense of events happening directly after each other (little physical room implying little room for mental closure) or even blending into each other.



Name: `close`
Dimensions: Inset ~1% of grid square (1 pixel on 60x60 square)
When used? Used to give the sense of events having closer temporal proximity, or more monotony to them, as panel closeness implies less room for mental closure.



Name: `normal`
Dimensions: Inset ~5% of grid square (3 pixels on 60x60 square)
When used? Used as default gutter size between most panels.



Name: `normplus`
Dimensions: Inset ~8% of grid square (5 pixels on 60x60 square)
When used? Slightly greater than `normal`; used as default gutter size between rows of panels to help guide the flow of reading. Placing between panels in middle of a row can isolate a panel (or group of panels) as if it is supposed to be read separately from the rest of the row. Could be used in conjunction with `roundrect` or

	`open` panels to subtlely set them apart further from the narrative stream of panels.
	**Name:** `wide` **Dimensions:** 1 grid squares **When used?** As mentioned in *External Panel Layout* section in the main document, used to set apart significant panels or imply an unmonitored gap of time.
	**Name:** `xwide` **Dimensions:** 2 or more grid squares **When used?** Used to set apart significant panels or imply a larger unmonitored gap of time (e.g. space between runs on a daily or monthly view). The larger the gutter the larger the gap of time or the greater the significance of the following or preceding panel. Could also be used to generate suspense, like slow motion, time dilation, and silence in film.

Appendix 3: Narrative Devices

The following table discusses several narrative devices that different panel shapes, sizes, and spacings can support. (The names for these narrative devices are arbitrarily given; there may be better, more appropriate, or more common names for these devices, but I am picking names here just for the sake of shorthand.)

There are so many ways to accomplish each of these narrative devices, that it is important that the authoring system be consistent—simply choose one of the listed approaches and choose it over others. I have tried to prioritise the execution solutions where possible below, but this prioritization is also biased. Different choices should still work, as long as consistency is maintained.

(Acronym glossary reminder: IED = *Improvised Explosive Device*, or *bomb*.)

Narr. Device	Criteria	Parameters	Execution
Flashes *(Jumps forward or backward in time, yet presented within present narrative.)*	Is there past or future information that *must* be presented in the context of the current narrative flow? On meeting a person (shaking hands) or sighting/ pursuing/capturing a suspect, has the user encountered this person previously?	• Relative importance of narrative events • Space available for panel placement • Panel from past narrative in which familiar person appeared? • Duration and number of panels from beginning of run's narrative to placement of *flash*	Use **roundrect** panel. May alternately use the "timeless" **open** panel, but **roundrect** is usually reserved for the thoughts (memories, as well as dreams and predictions) of the user's avatar. By and large, *flashes* will likely not be used in this implementation, because with short duration run narratives, past events could simply be put at the beginning of the narrative with an **xwide** gutter afterward, and the in-context *flash* would not be needed.
Loiters *(Temporal pauses or slowdowns, for slowing pace of story, or creating suspense.)*	Is a mundane activity followed by sudden unexpected activities? Is a decision (*a fork in the road*) being made by the avatar? Is a dramatic change in state about to occur?	• Relative importance of narrative events • Space available for panel placement	Use either **open** panel or **wide** or **xwide** gutter, or a combination. If enough space available, use the wide gutters; if space is a premium, then use the borderless open panel to provide same sense of timelessness. Wide gutters are better for suspense, since the lack of visual information gives the reader more time to invent possible panel closures in her mind.
Meanwhiles *(Simultaneous*	Is narrative using foreshadowing	• Relative importance of	Use **roundrect** panel. If roundrect style is already being

actions, most often in other locations.)	(e.g., IED narrative chain adds placement and other panels from adversary POV)? Is narrative tying into data from other wearable users in another location (at the same time)?	narrative events • Space available for panel placement	used for flashbacks within narrative of run, to avoid confusion, use thicker borders, **open** panels, or different border colors (with **rect** or **roundrect**) to distinguish. If panel sizes are the same, and geographical separation of *meanwhile* is vast, may also consider separating the group of panels by **wide/xwide** gutters on either side instead.
Grinds *(Long duration actions, often monotonous or generally uneventful.)*	Is there a long period of the same activity or series of expected and common activities?	• Relative importance of narrative events • Space available for panel placement	Use **wide** family of panels. Longer activities yield wider panels, but if space is unavailable, an **open** panel can create a similar sense of timelessness but does not have exactly the same kind of effect. Alternately, to save space, a series of **small** panels containing the same activity (identical or almost identical) creates the same effect. (Use at least 2 or 3 in sequence.) If panels surrounding *grind* are repetitions of the same series of activities (other grinds using small panels, perhaps), they may be absorbed into a larger **wide** panel with same effect and minimal information loss.
Bursts *(Sudden, temporally compressed, or emotionally intense actions.)*	Is there a relatively rare activity? Is there a short duration activity which is still important (or rare)?	• Relative importance of narrative events • Relative duration of rare activity • Space available for panel placement	Depending on relative narrative significance (high rarity and long duration), use a **blast** shape or **rect**. If **rect**, must vary size (as discussed below), but **blast** can also be used in combination with the following: **thin**, **large**, **xlarge**, or **tallthin.** Panels that are *bursts* but relatively short duration should use **thin**, or **tallthin**, if they involve vertical motion or relative height of panel subjects.
Forests *(Establishing shots or emphasis of scale.)*	Has the scene changed to a new location that the user will stay in for a relatively long duration in the narrative? Is there a person or	• Relative importance of narrative events • Space available for panel placement • Direction and speed of	For *forests* that are combinations with *loiters*, consider using the cinematic/panoramic **fullwide**. For panels that need to emphasize vertical motion or distance, use **tall** or **xtall,** depending on space available and magnitude of distance/movement. For panels that need to emphasize

	object approaching the user that should be noted? Is the user standing next to something very tall or very short that is involved in the narrative?	incoming item?	horizontal motion or distance, use **wide** or **xwide,** depending on space available and magnitude of distance/movement. For other panels, judiciously use **large**; may avoid confusion with narratively important **large** (rare, long duration) panels by making large panel slightly wider and/or combining with an **open** border style.
Twigs *(Highlighting significant details.)*	Is there an important detail in the panel (e.g., an IED)? Has the same activity occurred in several panels without change in location or introduction of unexpected activity?	• Relative importance of narrative events • Space available for panel placement • Location information for item to highlight (focus in upon)	Highlights may occur after or (in some cases) after a larger panel to make sure that the reader did not miss a certain detail. Use **circle**, possibly in conjunction with **large** if highlight is very rare. For zooms-in over repetitive activities, or in which rarity of highlight is not as great (relative to other twigs in run sequence), use **rect** instead. For extremely rare twigs, within run and within all panels for that user, may use **blast**.

Appendix 4: Activity Types

GPS is useful for all activities that involve movement over distance, particularly walking/running/driving/riding, as well as possibly being used to determine if the person is indoors/outdoors by location compared to a map. For direction, the compass can provide more context. The altimeters would be used for determining upstairs/ downstairs movement, but, depending on their resolution, they could possibly be used for assisting standing/kneeling distinctions. They might aid the sitting/riding distinction, should GPS not be working, since seats in cars are much higher off the ground than chairs at sea-level and much lower than second-story chairs.

Activities below in parentheses are currently not implemented in art assets.

Activity Type	Sensors	Code
walking	Accelerometers, GPS	w
walking_weapon	Accelerometers, GPS	W
(walking_upstairs)	Accelerometers, Altimeter, GPS	
(walking_upstairs_weapon)	Accelerometers, Altimeter, GPS	
(walking_downstairs)	Accelerometers, Altimeter, GPS	
(walking_downstairs_weapon)	Accelerometers, Altimeter, GPS	
standing	Accelerometers, GPS	s
standing_weapon	Accelerometers, GPS	S
running	Accelerometers, GPS	r
running_weapon	Accelerometers, Altimeter, GPS	R
(running_upstairs)	Accelerometers, Altimeter, GPS	
(running_upstairs_weapon)	Accelerometers, Altimeter, GPS	
(running_downstairs)	Accelerometers, Altimeter, GPS	
(running_downstairs_weapon)	Accelerometers, Altimeter, GPS	
(sitting)	Accelerometers, Altimeter, GPS	
driving	Accelerometers, Altimeter, GPS	u
riding	Accelerometers, Altimeter, GPS	v
riding_weapon	Accelerometers, Altimeter, GPS	V
standing_shakinghands	Accelerometers, GPS	h
(walking_shakinghands)	Accelerometers, GPS	
(kneeling_shakinghands)	Accelerometers, GPS	
kneeling	Accelerometers, Altimeter, GPS	k
kneeling_weapon	Accelerometers, Altimeter, GPS	K
(lyingprone)	Accelerometers, Altimeter, GPS	
(lyingprone_weapon)	Accelerometers, Altimeter, GPS	
(lyingprone_crawling)	Accelerometers, Altimeter, GPS	
(lyingprone_crawling_weapon)	Accelerometers, Altimeter, GPS	
standing_openingdoor	Accelerometers, GPS	d
standing_openingdoor_weapon	Accelerometers, GPS	D
(walking_openingdoor)	Accelerometers, GPS	
(kneeling_openingdoor_weapon)	Accelerometers, Altimeter, GPS	
(kneeling_openingdoor_weapon)	Accelerometers, Altimeter, GPS	

Appendix 5: Pseudocode walkthrough

Pseudocode for the basic prototype, assuming a mission type of *presence patrol*:

- Open the `activities.xml` file to start parsing
- Count the number of runs
- Create an array that will hold all runs
- For each run:
    - Count the number of activities in the run
    - Note the start time of the run
    - Note the amount of time between this run and the ones before it (if any) and after it (if any)
    - Create a Run object from the start time of the run and the number of activities that will hold all of the run information
        - Create an array that will hold activities
        - Create a string that will hold the single-letter codes for all of the run's activities
        - For each activity:
            - Note the duration both in the hours-minutes-seconds string that it comes in, and also calculating the total duration in seconds
            - Note the activity type *(see Appendix 4: Activity Types)*
            - Create an Activity object from the type and duration
                - Activity object will determine its own properties
                - Pick appropriate keyframe art for activity type; if activity type is *walking* or *standing* (e.g.), choose one of the multiple keyframes for that activity *(e.g., walking0, walking1, or walking2)*
                - Assign corresponding single-letter code for activity
                - Assign an arbitrary overall probability for that activity to occur *(this assumes that the mission type, e.g., presence patrol, is known…probabilities change between mission types)*
            - Put the Activity on the end of the run's array of activities
            - Put the single-letter code for the activity on the end of the string, followed by an integer representing the duration in seconds
        - Sum (and note) the total duration of the run
    - Put the Run on the end of the array of runs
- For each run in the array, put together the strings of letter-codes and duration-integers of each run into a big string, separated by a '%' character and an integer representing the duration between the runs on either side
- Create a Comics object from the start time of the first run and the total duration of the comics (all runs), with an array that will hold the panel descriptions

- Using regular expressions, search for unexpected activities following a story template (*in this case, for all following pseudocode, the* ***presence patrol*** *mission type, which primarily involves walking or riding to a place, walking around, shaking hands, kneeling, and standing*)
    - While there are still characters and digits in the string, search for activities that do not match one of the following contiguous patterns, starting with the first…
        1. *Intro phase:* walking, driving, or riding (to starting point)
        2. *Normal ops:* standing, walking, kneeling
        3. *Meeting people:* shaking hands
        4. *Entering/exiting a building:* opening door
        5. *Suspect sighting/pursuit:* any type of running, or any type of weapon combination activity (except for lying prone with weapon)
        6. *Sniper:* any type of lying prone, running, running with weapon, kneeling, or kneeling with weapon
        7. *IED (Improvised Explosive Device) discovery:* lying prone, lying prone with weapon, standing with weapon, standing, kneeling, kneeling with weapon, running (away)
        8. *Cordon and Search:* weapon raised while: standing, walking, running, kneeling, opening door, or going up- or downstairs
    - …if a non-greedy match is found for something NOT in that pattern (something unexpected—a state change, for example, from *Normal ops* to *Sniper*), then
        - Take the first part of the string that matched the expected activities and pass it to a makePanels method of the Comics object along with the assessed state
        - Trim that part (the expected activities) off the beginning of the string and make the string equal to the resulting substring
        - Switch statement of expected activities
            - If previously 1, then follow by looking for 2
            - If previously 2, then follow by looking for 3 (see next)
            - If previously 3, 4, 5, 6, or 7, then follow by looking for 2, and failing that, the next in series (for 3: 4 then 5 then 6 then 7 then 8)
                - Sidenote: expect 2 to be, as its name "*Normal ops*" implies, the default set of activities that bookends each change of state…this will not *always* be the case, as with a *IED discovery* state that changes into a *Suspect sighting/pursuit* state, but it is a good rule of thumb
            - If previously 8 (no success after searching series), look for 2
            - If, at this point, no activity pattern from within 2-8 is found, then isolate the string of activities between the start of the

string and the beginning of a state 2 or the end of the string and mark it as a state 0 (unknown)
        - Example: driving a vehicle fits none of these state patterns, and would possibly be rather rare in the *middle* of a presence patrol
    - Recurse and do everything in this parent step again while there are still letters and digits in the string
- The Comics makePanels method should
    - Take in the state and the activity/duration string so that they may be used create panels, for the eventual XML output, within an array property of the Comics object
    - For each activity, put together a panel description (*either a Panel object with properties or an XML string within an array, following the specification in the main document above*) using the default panel size, shape, and gutter
        - A panel should have the following properties
            - A relative size (Appendix 2)
            - Border style (Appendix 2), thickness, and color
            - Gutter sizes (Appendix 2) before and after the panel
            - Optionally, a subject (by default, the user's avatar)
            - An activity keyframe for this subject (Appendix 4)
            - Optionally, an object of that activity (object art assets will automatically be used for shaking hands , driving, riding, or opening a door, for example, but there is also the option of changing this object, e.g., driving a sports car instead of a humvee, or of adding a character that is the object of another action, e.g., showing the suspect that the user's avatar is holding at gunpoint)
            - Optionally, a composition zoom level (defaults to either macro or mono, more info below)
            - Optionally, in future implementations, further metadata (which could help in rendering the panel background or the characters)
                - GPS location and compass orientation (in panel description instead of as a later separate query, for purposes of drawing background buildings from a 3d map of the area)
                - Historical weather data for that time and location
        - Panel defaults (for size, border, gutter, and compositional zoom level) should be already specified in the code (recommend: *normal*, *norm*, *normal*, and *macro*, respectively)

- Rank panels' importance by looking at the duration of the activity (or, grouping the activities, the state) with respect to the duration of the run, and the rarity of the activity with respect to the other activities in the run as well as to the likelihood of such an activity based on the mission type
    - Importance ranking will work alongside and within state classification to help differentiate how to style panels and how to highlight "expected" vs. "unexpected" activities for summarization (noting what should be kept in when the timescale is expanded)
    - Recommend equation of

$$\text{importance} = \left(\begin{matrix}\text{duration}\\ \text{of activity}\end{matrix}\right) \log \frac{\left(\begin{matrix}\textit{presence}\\ \textit{patrol}\\ \text{weighting}\\ \text{coefficient}\end{matrix}\right)\left(\begin{matrix}\text{probability}\\ \text{that this state}\\ \text{is in presence}\\ \text{patrol}\end{matrix} + \begin{matrix}\text{probability}\\ \text{that this}\\ \text{activity is in}\\ \text{presence}\\ \text{patrol}\end{matrix} + \begin{matrix}\text{probability}\\ \text{that this}\\ \text{activity is in}\\ \text{this run}\end{matrix}\right)}{\left(\dfrac{\text{duration of state}}{\text{duration of run}}\right)}$$

    modified slightly from the importance equations of FXPAL's Video Manga project (Uchihashi, et al. 1999)
- *Take out panels:* Consolidate similar or repetitive activities based on their inherent probabilities (assigned on Activity instantiation arbitrarily with respect to mission type)
    - For common, high probability activities, such as walking/standing/kneeling:
        - If the activity is extremely brief on the timescale of the run itself, and it is between activities that are of much longer duration, then it can be omitted from the summary
        - If the activity is surrounded by activities of similar type, rarity, and duration, they can
            - be represented as a series of very small panels for each activity
            - be lumped into one *grind* panel (see Appendix 3 above)
    - For less common activities for the mission type, that are common in the run (e.g., a long period of alternating running with and without weapon raised, punctuated by a few very brief moments of standing to get situational awareness) the activities can be consolidated into one or two *grind* panels of running
- *Add panels:* Where narrative necessitates, add panels or style panels to provide better narrative flow
    - Add new panels
        - If a *Sniper* or *IED discovery* state is in the run, put in *flash* panel (or panels) at beginning of run (or after

establishing shots, see below) to set up, showing the IED being placed or the sniper in location, making sure to designate that the subject of the panel be a character who is not the user's avatar
    - If (in future) adding in concurrent run data from other wearable users, add salient *meanwhile* panels in the narrative at right temporal points
- Style existing panels
    - If GPS or activity sequence (e.g., running, walking, driving, or riding for a relatively long duration and then standing) shows arrival in a new area, make this first panel in the new area a *forest* panel to establish the location (always provide a *forest* after the state change from 1 to 2 at beginning of run)
    - If activity is going up or down stairs, make the panel size *tall* (or within the *tall* family)
    - If the delta between panel states or importance levels is significant (e.g., *Normal ops* to *Sniper*), make the first panel of the new state a *blast* …but keep in mind that this panel shape should NOT be overused (see caveats in Appendix 2)
    - In *grinds*, and in other situations where multiple panels are used and the activities alternate back and forth between only two activities (within the same state), the "camera" of the panel should slowly zoom in on the subject, moving from Cohn's *macro* to *mono* (or, in future implementations, *micro*, the extreme close-up)
        - Purpose of this slow zoom is to provide a more natural gradation of zoom levels that is more pleasing to the eye and the read-flow than simply "near" and "far" as in Comic Chat (Kurlander, et al. 1996)
        - Currently, zoom level is interpolated in the render stage; in future implementations, it seems better to put this in the actual panel description XML

- When the processing of the activities string is complete, and makePanels has been used to create panel descriptions for the entire comics narrative, write out a complete XML file (see *XML Format for Comics and Panel Descriptions* in main document above)
    - Include proper headers, along with identifying metadata culled from the user's profile (e.g., name, contact information) and the

Comic itself (e.g., start time, end time), as well as the arbitrarily hard-coded defaults (e.g., default panel size)
    - Within a *panels* element, write a *panel* element for each panel with the following where it changes from the defaults:
        - *size* element with a general size inside
        - *border* element with style, thickness, and color
        - *gutter* element with the gutter sizes before and after the panel
        - *panel-desc* element with
            - an *action* element describing the activity keyframe and, optionally, the subject and object of the action as attributes
            - a *composition* element describing the zoom level
    - Close any open XML elements and finish XML document
- The Flash file should take in the resulting XML file and render it into comics panels
    - Take in the XML file using Flash's built in XML object
    - Make a new Comic object
        - Has default properties of an (x,y) location on the screen, screen dimensions (default: 600 x 360 for our proportions), and the default square grid dimensions (60x60-pixel square, 10x6 squares)
        - Has array of Panel objects
        - Creates two nested MovieClips, one for content and another above it for masking that content; content MovieClip can be moved behind the clipping mask to give scrolling effect (*more detail to be given below when discussing creation of Panel MovieClips*)
    - Loop through the resulting XML node-tree and create Panel objects from the *panel* elements in the XML (passing all of the attributes in the XML to the Panel constructor)
        - Panel size, border, and spacing should carry over directly
        - For *activity*, choose the appropriately-titled keyframe from the subject's MovieClip (e.g., the keyframe "*walking*2" for the default subject "*soldier*" …for more information, double-click on the *soldier* MovieClip in the *p4.fla* Library within Flash to see all of the keyframes available)
        - If there is an object, choose an appropriate keyframe from the "*object*" MovieClip (*currently does not exist, as this is not presently implemented*)
    - Create a MovieClip for the Panel (*panelHolder*) within which other MovieClips will be nested
        - Create a MovieClip for the content, within which all art assets within the panel will be drawn

        - Create a MovieClip for the clipping mask (a solid-filled shape matching the panel's shape and border) on a layer above the content
        - [*Note: When creating MovieClips in code, there are no layers akin to the layers in the Flash timeline in the main UI; all MovieClips are placed in the same root MovieClip, so when referring to Panels within the array in Comic, make sure you type the correct path to your MovieClip to avoid accidentally overwriting content*]
    - If panel packing is to be done, it should be done here before the panel is placed and the content is scaled and drawn
        - Basic algorithm for panel packing, also drawn from the Video Manga work (see citation below):
            - Fill each row with panels via brute force calculation
                - Set the starting panel to the first panel
                - For each row height from 1 to 6 grid squares:
                - Find all the panel sequences, from the starting panel to panel *n*, that will fit the row block of this row height
                - Some row heights will be impossible to fit without shrinking the panel height; consider this in cases where shrinking the panel height will not change its size classification (e.g., changing a 2x5 *xtall* panel to a 2x4 *xtall* panel still has the same effect; see Appendix 2)
                - Stack panels left to right and then top to bottom inside row
                - Note how much space is remaining
                - Note whether panel size or spacing of each panel in the sequence up to panel *n* can be adjusted while still keeping same relationship between panels; there are ranges of values rather than specific values on many of the panel variations and narrative devices for this exact purpose (see Appendices 2 & 3)
                - Mark the best fit
                - Note the positions of the panels in the best fit for this row block
                - Start a new row (set the starting panel to *n*) and repeat parent step until all panels in Comic are placed
            - Mark row divisions so that gutters between rows can be set accordingly

        - FXPAL's Video Manga team discusses Panel Packing in full (Uchihashi & Foote 1999); while their their panel packing outlined here does not use panels with height or width that vary from their standard 4:3 proportions, the logic behind it should work for panels of different sizes as well
    - Draw the content of the panel
        - Determine the compositional zoom level from the XML and scale the content accordingly
            - [*Note: Flash requires you to scale before you draw, but scaling changes the coordinate system as well; therefore always: move, scale, draw*]
            - For *macro*, the subject and object asset characters should be scaled so that their full height and width fit within the width of the panel
                - The panel composition should center on a point at the center of a line drawn between the center of the two characters (or assets)
                - The scale should be set so that the head and feet of the character art fits within a 5 pixel margin of the border of the panel
            - For *mono*, the subject character should be scaled so that its width fits within the width of the panel, and the top of the head should be inset a margin of 5 pixels from the top of the panel (or further—25 pixels if the panel is a *blast* type panel, which generally has an inset of 20 pixels for its spikes)
        - Draw a background
            - Assets should be scaled to match subject/object
            - Horizon line should stay consistent; should be placed between bottom of feet and middle (waist) of standing character art
            - Place sky, 2D representations of buildings, background characters, and props
                - Ideally, *Entering/exiting building* states, combined with GPS location and a map, should be able to tell the software whether or not the user is currently indoors or outdoors, which would affect the background drawn; different classifications of building (e.g., hut, cormex [a large shipping container with windows and electricity used as housing by deployed military], house, office, factory, etc.) could change the background style in this case
                - Sky should be colored based upon time of day and historical weather data; this information would also affect consistent rendering of other assets in panel

        - Ideally, building representations should be drawn based on the location and orientation of the wearer, from the perspective of the virtual camera; simplistic buildings (rectangles) can be drawn and sized corresponding to wearable user's proximity to buildings on map as shown by GPS data
        - Should ASSIST analysis of audio and images (timestamped within the duration of this panel) be able to identify certain elements nearby at that time (vehicles, animals, crowds of people) these assets can be placed in background of panel frame
    - Exact positioning of background elements will not be accurate; either…
        - refine this in future implementations, following example of Comic Chat algorithms (Kurlander, et al. 1996)
        - allow user to manipulate (e.g., drag and drop) relative positions in user interface as an annotation
    - Background should be detailed, but not photographic, because the stylistic mismatch would be jarring to the reader and the level of detail would probably slow the eye movement on the page
- Draw the panel's mask and border
    - Create an array of coordinates that will contain the points of the border (examples follow)
        - For a normal rectangular border, add four sets of screen coordinates for each of the four corners of the panel
            - Factor in the grid square location, size, and gutter size to calculate the corner coordinates
            - For gutters, do not forget to inset the panel corners as many pixels as warranted (see Appendix 2)
        - For a *roundrect* panel, add two sets of coordinates along rectangle before and after each corner; these will be used for drawing Bezier curves with the corner coordinates as control points
        - For a *blast* panel,
            - add several points along the rectangle edges between the corners
            - add points between each of the existing points, inset by about 10% of the panel height/width
            - vary the inset of these points by a few pixels randomly for a more natural look
            - vary spacing of these points sinusoidally (see Appendix 7 for an example) so that points are closer

            together in the middle and further apart toward the corners, so that the spikes do not overlap in corners, causing strange drawing artifacts
        - Connect the points to create a solid-filled shape with no line border in the mask MovieClip
        - Draw the border in the *panelHolder* MovieClip for this Panel, by connecting the points to draw an unfilled shape with a line border matching the color and thickness from the XML
    - Add event handler callback function to the Panel so that when it is clicked, it will
        - highlight the panel, by drawing a translucent yellow rectangle behind the panel
        - call up any associated annotations from a text file or database
        - query a directory (or a database of filenames) to find associated media related to the time period of that panel so that icon links for the media can be created (see following bullet and Appendix 8)
- Once Comic has been rendered, build a user interface (UI) around it (as outlined in Appendix 8)
    - Create scroll buttons on the right side of the screen
        - Scroll comic up or down by rolling over buttons
        - Scrolling accomplished by moving the Comic's content layer up or down while keeping the Comic's mask position static on the screen
    - As specified above, on clicking a panel, bring up any annotations in lower left pane and any iconic links to associated media (filename fitting within the datetime range of the panel, including before and after gutters) in lower right
        - Add onClick event handler to annotation pane, so that it transforms into a text input field (shows a hidden MovieClip) which allows user to add or edit a timestamped annotation for that comics panel; annotations submitted when submit button is clicked
        - When Panel is highlighted, show icons for audio/video/images in lower right
            - Rendering of this icon bar will be similar to rendering of Comic, with a MovieClip containing nested content and mask MovieClips
            - Add scroll buttons (with rollover event handlers) if the total width of the icons exceeds the space available (1/2 the width of the screen)
            - Add onClick handler to the icons so that clicked icons are dimmed to show that they have been visited, and content for that icon is loaded in a separate window
    - Place map MovieClip in upper left corner of screen to provide location context for any highlighted Comic panels
        - Add event handlers to map:

            - Rollover will show a plus symbol (+) in the lower left implying that it can be clicked
            - Clicking anywhere on the mini-map will double its size, expanding it to a much larger, translucent (so that Comic underneath can be seen) map of the area, with the current location of the wearable user shown if a Panel is highlighted
            - Rollover of the map will now show a minus symbol (-) in the lower left corner of the large map, implying that it can be collapsed
            - Clicking this minus symbol will shrink the map to its original (mini-map) size
    - Draw timeline MovieClip along top of screen
        - Draw markers appropriate to timescale
            - A typical run will only go for hours, not days
            - If there is less than an hour in the set of runs, number every 10 minutes and put hashmarks every 5 minutes
            - Else if there is less than four hours in the set of runs, number every 30 minutes and put hashmarks every 15 minutes
            - Else if there is less than a day in the set of runs, number every 4 hours and put hashmarks every 1 hour
            - Else if there is less than a month in the set of runs, number every 7 days and put hashmarks every 1 day
            - Else if there is less than one year in the set of runs, number every four months and put hashmarks every month
            - Longer than one year should probably be split up, as using a relatively tiny timeline to visualize navigate such a large space of data is probably not very effective
        - While not changing length of timeline bar, add drag event handlers to allow user to drag at ends of it to adjust timescale
            - By changing timescale on timeline, user could
                - change the default size of squares on the grid to make smaller panels (a zoomed-out page view of sorts…although the effectiveness of this view would have to be evaluated with testing)
                - change the number of panels generated for the run, allowing panels to be omitted as the timescale zooms out (for covering a longer period of time); this would require a new panel importance threshold to be set, and the Comic to be re-rendered
        - Add drag event handlers to inner rectangle in timeline that shows currently highlighted panels (or currently visible panels) as alternate method of scrolling and selecting the Comic

Appendix 6: PHP Source Code

The purpose of the PHP code was, as specified earlier, to parse the activities XML into comics description XML. There are two php classes, Activity (activity.php) and Run (run.php). A Run object contains an array of Activity objects. Runs of activities are drawn from the activities.xml file, which would be generated from the raw data by the pattern-matching, hidden Markov model algorithms of the Contextual Computing Group.

activity.php

Activity describes an activity taken from a line in the activities.xml. Its constructor takes five parameters: the activity type (see *Appendix 4: Activity Types* above), the total duration in seconds, and its constituent hours, minutes, and seconds. It takes the activity type and determines the Flash keyframe required for that activity (since some activities, such as shaking hands and opening doors, use the same keyframe) as well as the single letter abbreviation for that activity and the (in this demonstration case) arbitrarily assigned probability that such an activity would occur in a presence patrol. Finally, it provides some encapsulation functions with abbreviated single-letter names for its methods and properties.

```
<?php
class Activity {

      var $type;
      var $duration;
      var $dura_hrs;
      var $dura_min;
      var $dura_sec;
      var $letter;
      var $probability;
      var $keyframe;
      var $frac_run;

      function Activity($t,$h,$m,$s,$d){
            $this->type = $t;
            $this->duration = $d;
            $this->dura_hrs = $h;
            $this->dura_min = $m;
            $this->dura_sec = $s;

            switch($t) {
                  case("walking"):
                        $this->letter = "w";
                        $this->keyframe = $t;
                        $this->probability = 25;
                        break;
                  case("walking_weapon"):
```

```
                        $this->letter = "W";
                        $this->keyframe = $t;
                        $this->probability = 3;
                        break;
                  case("standing"):
                        $this->letter = "s";
                        $this->keyframe = $t .
floor(rand(0,2.999));
                        $this->probability = 20;
                        break;
                  case("standing_weapon"):
                        $this->letter = "S";
                        $this->keyframe = $t;
                        $this->probability = 3;
                        break;
                  case("running"):
                        $this->letter = "r";
                        $this->keyframe = $t;
                        $this->probability = 4;
                        break;
                  case("running_weapon"):
                        $this->letter = "R";
                        $this->keyframe = $t;
                        $this->probability = 1;
                        break;
                  case("driving"):
                        $this->letter = "u";
                        $this->keyframe = $t;
                        $this->probability = 4;
                        break;
                  case("riding"):
                        $this->letter = "v";
                        $this->keyframe = $t;
                        $this->probability = 5;
                        break;
                  case("riding_weapon"):
                        $this->letter = "V";
                        $this->keyframe = $t;
                        $this->probability = 1;
                        break;
                  case("shakinghands"):
                        $this->letter = "h";
                        $this->keyframe = "standing_door_shake";
                        $this->probability = 7;
                        break;
                  case("kneeling"):
                        $this->letter = "k";
```

```
                        $this->keyframe = "knee" .
floor(rand(1,2.999));
                        $this->probability = 15;
                        break;
                  case("kneeling_weapon"):
                        $this->letter = "K";
                        $this->keyframe = $t;
                        $this->probability = 3;
                        break;
                  case("openingdoor"):
                        $this->letter = "d";
                        $this->keyframe = "standing_door_shake";
                        $this->probability = 8;
                        break;
                  case("openingdoor_weapon"):
                        $this->letter = "D";
                        $this->keyframe = "standing_weapon";
                        $this->probability = 1;
                        break;
                  default:
                        $this->letter = "Q";
            }
      }

      function d(){
            return $this->duration;
      }

      function k(){
            return $this->keyframe;
      }

      function setd($number) {
            $this->duration = $number;
      }

      function t() {
            return $this->type;
      }

      function l() {
            return $this->letter;
      }

      function p() {
            return $this->probability;
      }
```

```
        function h() {
              return $this->dura_hrs;
        }

        function m() {
              return $this->dura_min;
        }

        function s() {
              return $this->dura_sec;
        }

        function f() {
              if (isset($this->frac_run)) return $this->frac_run;
        }

        function setf($rund) {
              $this->frac_run = $this->duration / $rund;
        }

  }

  ?>
```

run.php

A Run object is constructed with three parameters: an associative array containing the start time (in string form, and in year, month, day, hour, minutes, and seconds constituents), an array of the Activity objects within that run, and a total duration in seconds. Runs will likely only be a few hours a piece; it is unlikely that runs (continuous collections of data) should exceed 9 hours, so tracking the duration in seconds should not run into number-too-large issues (such as exceeding the value of an integer or getting into floating-point discrepancies).

```
  <?php
  class Run {

        var $start;
        var $activities;
        var $duration;
        var $al;

        function Run($st, $ac, $d) {
              $this->start = $st;
              $this->activities = $ac;
              $this->duration = $d;
              $this->al = count($ac);
        }
```

```
        // 'fullstr'=>$run_matches[1][$i],
        //  'year'=>$start_parts[0],
        //  'month'=>$start_parts[1],
        //  'day'=>$start_parts[2],
        //  'hour'=>$time_parts[0],
        //  'min'=>$time_parts[1],
        //  'sec'=>$time_parts[2]),

        function st_f(){
              return $this->start['fullstr'];
        }

        function st_yr(){
              return $this->start['year'];
        }

        function st_mo(){
              return $this->start['month'];
        }

        function st_da(){
              return $this->start['day'];
        }

        function st_h(){
              return $this->start['hour'];
        }

        function st_m(){
              return $this->start['min'];
        }

        function st_s(){
              return $this->start['sec'];
        }

        function d(){
              return $this->duration;
        }

        function a($i) {
              return $this->activities[$i];
        }

}

?>
```

comic.php

Since PHP can output any type of text or binary data, as long as it sets the HTTP headers before writing, we can use it to write XML by specifying

```
//header("Content-type: text/xml; charset=UTF-8");
```

before the code proper. This PHP file was eventually supposed to generate XML when called, so that it would be used as the comics description URL parsed in Flash. Currently, it is an early (deprecated) version of a test file to see if it can properly parse activities.xml. This functionality had been successfully made object-oriented in testbed.php, but had not yet been moved back into comic.php.

```
<?php
// need to do this first
//header("Content-type: text/xml; charset=UTF-8");
include "activity.php";
include "run.php";

$xml = file_get_contents("activities.xml");
echo "$xml\n\n\n";
preg_match_all('#<run
start="(.+?)">(.+?)</run>#s',$xml,$run_matches);
$runlist = array();

for($i=0;$i<count($run_matches[2]);$i++) {

      $start_parts = explode("-",$run_matches[1][$i]);
      $time_parts = explode(".",$start_parts[3]);
      $time_parts[2] = (isset($time_parts[3])) ?
$time_parts[2].'.'.$time_parts[3] : $time_parts[2];

      preg_match_all('#<activity type="(.+?)" dura="(.+?)" />#',
$run_matches[2][$i], $activity_matches);
      $activities = array();
      $durations = array();
      for($j=0; $j<count($activity_matches[1]); $j++) {
            $dura_parts = explode(".",$activity_matches[2][$j]);
            $dura_parts[2] = (isset($dura_parts[3])) ?
$dura_parts[2].'.'.$dura_parts[3] : $dura_parts[2];
            $durations[$j]= $dura_parts[0]*3600
+$dura_parts[1]*60+$dura_parts[2];

            $activities[] = new
Activity($activity_matches[1][$j],

$dura_parts[0],$dura_parts[1],$dura_parts[2],

$durations[$j]);
      }
```

```
      $runlist[] = new Run(array('fullstr'=>$run_matches[1][$i],

'year'=>$start_parts[0],

'month'=>$start_parts[1],

'day'=>$start_parts[2],

'hour'=>$time_parts[0],

'min'=>$time_parts[1],

'sec'=>$time_parts[2]),
                                    $activities,
                                    array_sum($durations) );
}

$sillystring = "";
for($i = 0; $i < count($runlist); $i++) {
      for($k = 0; $k < $runlist[$i]->al; $k++) {
            $a = $runlist[$i]->a($k);
            $runlist[$i]->activities[$k]->setf($runlist[$i]-
>d());
            $sillystring .= $a->l().$a->d();
      }
}

echo '<pre>';
var_dump($runlist);
echo '</pre>';
echo "\n\n$sillystring";


//function makeComic($title,$date,$descrip,$url,$dw,$dh,$gl,$gr){
function makeComic($panels) {
      echo '<'.'?xml version="1.0" encoding="utf-8"?'.">\n";
?>
<comic>
      <title>test comic</title>
      <last-built>2006-03-22 02:07:02</last-built>
      <description> blah</description>
      <url>comic.xml</url>
      <panelsize-default>
            <width>120</width>
            <height>90</height>
            <gutter-left>10</gutter-left>
            <gutter-right>10</gutter-right>
          <border>
```

```
            <shape>normal</shape>
            <thickness>3</thickness>
            <color>0x000000</color>
          </border>
          <type>macro</type>
      </panelsize-default>
      <panels>
<?php
      foreach ($panels as $panel) {
            makePanel($panel);
      }
      echo "\t</panels>\n</comic>";

}

function makePanel($panel){

}



?>
```

testbed.php

This PHP file was meant to be a testbed for code before moving the finished product into comic.php. Neither testbed nor comic.php correctly identify strings of activities in the way intended, using their regular expressions.

```
<?php

include "activity.php";

$a = new Activity("standing", 2,4,3,5);
$a->setd(256);
echo "duration ".$a->d()."<br /> \n";
echo "letter ".$a->l()."<br /> \n";
echo "keyframe ".$a->k()."<br /> \n";
echo "probability ".$a->p()."<br /> \n";

$sillystring = "wwSKsk";
echo "$sillystring <br />\n";



$isStoryDone = false;
$presence = 0;
$suspect = 0;
$sniper = 0;
$doorshake = 0;
```

```
$durPresencePatrol = array();
$durSuspect = array();
$durSniper = array();
$durDoorshake = array();

$count = 0;
while(!$isStoryDone) {
	$count++;
	echo $count."<br />";
	if ($count==6) {
		$isStoryDone = true;
	}
}

/*
function findStory($actys, $st, $i){
	$s = ($i > 0) ? substr($st,$i) : $st;

	if (preg_match_all('#[^'.$actys.']#',$s,$matches) > 0) {
		echo "found something unusual<br />\n";

		var_dump($matches);
		foreach($matches[0] as $match){
			echo " ".$match;
			echo " ".strpos($s, $match);
		}
		$pos = strpos($s, $matches[0][0]);

		switch($actys){
			case "wsk":
				$durPresencePatrol[] = $pos;
				$presence++;
				//echo " // found something unusual at
$pos of $st - what is it? // ";
				break;
			case "SRrWD":
				$durSuspect[] = $pos;
				$suspect++;
				//findStory("wsk", $s, $pos);
				break;
			case "KkRr":
				$durSniper[] = $pos;
				$sniper++;
				//findStory("wsk", $s, $pos);
				break;
			case "hd":
				$durDoorshake[] = $pos;
				$doorshake++;
```

```
                        //findStory("wsk", $s, $pos);
                        break;
                }

                switch ($matches[0][0]){
                                case "h";
                                case "d";
                                      findStory("hd", $st, $pos);
                                      break;
                                case "S":
                                case "R":
                                case "W":
                                case "D":
                                      findStory("SRrWD", $st,
$pos);
                                      break;
                                case "K":
                                case "r":
                                      findStory("KkRr", $st, $pos);
                                      break;
                                case "w":
                                case "s":
                                case "k":
                                      findStory("wks", $st, $pos);
                                      break;
                }
                return;

        } else {
                echo "yes, it finishes.";
                switch($actys){
                        case "wsk":
                                $durPresencePatrol[] = strlen($s);
                                $presence++;
                                break;
                        case "SRrWD":
                                $durSuspect[] = strlen($s);
                                $suspect++;
                                break;
                        case "KkRr":
                                $durSniper[] = strlen($s);
                                $sniper++;
                                break;
                        case "hd":
                                $durDoorshake[] = strlen($s);
                                $doorshake++;
                                break;
                }
```

```
            return;
      }
}

findStory("wsk",$sillystring, 0);
*/

?>
```

Appendix 7: Flash ActionScript Source Code

Attached is the relatively context-free Flash source code for the currently not-quite-working prototype. The full Flash files will be available for download at http://idt.gatech.edu/ms_projects/jalderman. In addition to these ActionScripts, those files contain important MovieClips that are used in the creation of a comics summary.

Panel.as

The Panel object is created for each panel in the comic. It controls the properties and drawing of the panel. This code is not satisfactory; it does not currently implement the grid algorithm, nor assist with panel packing (Uchihashi & Foote 1999).

```
class Panel {
      var _x:Number, _y:Number, _id:Number;  // x, y
      var importance_rank:Number;
      var blocks_wide:Number, blocks_high:Number;
      var blocksize:Number;
      var _height:Number, _width:Number;
      var gutter_before:Number, gutter_after:Number;
      var border_shape:String, border_thickness:Number,
border_color:Color, border_style:String;
      var border_points:Array;

      var pHolder:MovieClip;

      // n:String  ...  _root[n].create...
      public function Panel(x:Number, y:Number, id:Number,
d:Number, n:String, bs:Number) {
            this.pHolder =
_root[n].cCanvas.createEmptyMovieClip("pHolder"+id, d);
            this._id = id;
            this.blocksize = bs;
            var pContent:MovieClip =
pHolder.createEmptyMovieClip("pContent", 1);
            var pMask:MovieClip =
pHolder.createEmptyMovieClip("pMask", 1000);
            this._x = this.pHolder._x = x;
            this._y = this.pHolder._y = y;
            trace("Created a panel.");
            trace("It has a movieClip "+this.pHolder._name+" that
has movieClips: "+this.pHolder.pMask._name);
            trace("It is at "+this._x+", "+this._y);
            border_points = new Array();
      }

      public function setBorder(s:String, th:Number, col:Color,
sty:String):Void {
```

```
            this.border_shape = s;
            this.border_thickness = th;
            this.border_color = col;
            this.border_style = sty;
            trace("set the border shape " + this.border_shape +
", thickness " + this.border_thickness + ", color " +
this.border_color + ", style " + this.border_style);
      }

      public function setDimensions(w:Number, h:Number):Void {
            this._height = h;
            this._width = w;
            var cr = 20;  // corner radius for rounded rectangle
shaped panels
            switch(this.border_shape) {
             case "circle":
             case "circlebleed":
                  var steps:Number = (w+h)/3;
                  var offset_x:Number = w/2;
                  var offset_y:Number     = h/2;
                  for (var i:Number = 0; i < steps; i++) {
                        this.border_points[i] = {x:
w/2*Math.sin(i * 2 * Math.PI/steps)+offset_x , y: h/2*Math.cos(i
* 2 * Math.PI/steps) + offset_y};
                  }
             break;
             case "normal":
             case "rect":
             case "bleed" :
                  this.border_points[0] = {x: 0, y: 0};
                  this.border_points[1] = {x: this._width, y: 0};
                  this.border_points[2] = {x: this._width, y:
this._height};
                  this.border_points[3] = {x: 0, y:
this._height};
                  break;
             case "roundrect":
                this.border_points[0] = {x: 0, y: 0};
                  this.border_points[1] = {x: this._width, y: 0};
                  this.border_points[2] = {x: this._width, y:
this._height};
                  this.border_points[3] = {x: 0, y:
this._height};
                  // this is horribly inelegant.  i need to fix
this.
                  // the fact that AS only allows ONE element of
the array to be added at one time...
                  this.border_points.splice(4,0,{x:0,y: 0+cr});
```

```
                this.border_points.splice(4,0,{x:0,y: h-cr});
                this.border_points.splice(3,0,{x:0+cr,y:h});
                this.border_points.splice(3,0,{x:w-cr,y:h});
                this.border_points.splice(2,0,{x:w,y:h-cr});
                this.border_points.splice(2,0,{x:w,y:0+cr});
                this.border_points.splice(1,0,{x:w-cr,y:0});
                this.border_points.splice(1,0,{x:0+cr,y:0});
                break;
            case "ripply":
            case "blast":
            case "thought":
                this.border_points.push({x:0,y:0});
                addjagpoints('x',0,this._width,0,-
20,Math.floor(this._width/30));
                this.border_points.push({x:this._width,y:0});

      addjagpoints('y',0,this._height,this._width,20,Math.floor(t
his._height/30));

      this.border_points.push({x:this._width,y:this._height});

      addjagpoints('x',this._width,0,this._height,20,Math.floor(t
his._width/30));
                this.border_points.push({x:0,y:this._height});
                addjagpoints('y',this._height,0,0,-
20,Math.floor(this._height/30));
                break;
            }
      }

      private function addjagpoints(ax:String, s0:Number,
sn:Number, cv:Number, mg:Number, pt:Number) {
            // ax = axis of the new additions (e.g., if along x
axis, y will be constant, cv)
            // s0 = start value (e.g., x = 0)
            // sn = end value (e.g. x = 25)
            // cv = constant value for other axis (e.g. y = 0)
            // mg = margin, or how much inset to allow for
instrokes (must be +/- depending)
            // pt = how many points, excepting corners, will be
on the panel-border-line
            var xarr:Array = new Array();
            var yarr:Array = new Array();
            var i:Number;
            var pta:Number = (pt*2)+1;       // pta = number of
points to add total
            var db:Number = (sn-s0)/(pta+1); // db = avg dist
between points
```

```
            for (i=0; i<pta; i++) {
                  if (ax == 'x') {
                        //yarr[i] = (i%2 == 1) ? cv : cv-
(mg+(0.2*mg)-((0.4*mg)*Math.random()));
                        yarr[i] = (i%2 == 1) ? cv : cv-
((mg+((0.2*mg)-(0.4*mg*Math.random()))) * -0.5*(Math.cos((i/(pta-
1))*0.8*Math.PI+0.65*Math.PI)-1));
                        //xarr[i] = s0 + (i+1)*db;
                        xarr[i] = s0 + (i+1)*db+(-
.9*db)*Math.cos((i/pta-1)*Math.PI);
                        trace(-Math.cos((i/pta-1)*Math.PI));
                  } else if (ax == 'y') {
                        //xarr[i] = (i%2 == 1) ? cv : (cv-
(mg+(0.2*mg)-((0.4*mg)*Math.random())));
                        xarr[i] = (i%2 == 1) ? cv : cv-
((mg+((0.2*mg)-(0.4*mg*Math.random()))) * -0.5*(Math.cos((i/(pta-
1))*0.8*Math.PI+0.65*Math.PI)-1));
                        //yarr[i] = s0 + ((i+1)*db);
                        yarr[i] = s0 + (i+1)*db+(-
0.9*db)*Math.cos((i/pta-1)*Math.PI);
                  } else {
                        return;
                  }
                  this.border_points.push({x:xarr[i],y:yarr[i]});
                  //trace("adding x: "+xarr[i]+ " and y:
"+yarr[i]);
            }
            return;
      };

      public function createContent(s:Number, a:String,
t:String):Void {
            // s is the number of the panel asset to create
            // a is the action of the soldier
            if (a==null) a = "standing";
            if (t==null) t = "mono";
            // t is the macro / mono / micro string...if it's
macro, it's "macro"; mono, ""; micro, name of target?
            // CREATE BACKGROUND
            // this part needs tweaking...and the image should be
replaced by a vector of horizon line + bldgs

      this.pHolder.pContent.createEmptyMovieClip("image_"+s,
300);  // 300 is a random depth i picked

      this.pHolder.pContent["image_"+s].loadMovie("iraq.jpg");
            this.pHolder.pContent["image_"+s]._x = 0-(s*10);
```

```
            this.pHolder.pContent["image_"+s]._y = (this._height
> 90) ? -20 : this._height - 150;

      trace("****************panel"+s+"***"+this._y+","+this._hei
ght+","+this.pHolder.pContent["image_"+s]._height);
            with(this.pHolder.pContent["image_"+s]){

      trace("****************x:"+_x+",y:"+_y+",w:"+_width);
            }
            this.pHolder.pContent["image_"+s]._xscale = 50;
            this.pHolder.pContent["image_"+s]._yscale = 50;
            this.pHolder.pContent["image_"+s]._alpha = 50;
            //trace("created panel background");

            // CREATE THE SOLDIER
            var shakinghands = false;

      this.pHolder.pContent.attachMovie("soldier","soldier_"+s,
500);
            trace("soldier_"+s+" = "+
this.pHolder.pContent["soldier_"+s]._name);
            if (a == "shakinghands") {
                  shakinghands = true;
                  a = "standing_door_shake";
            }
            this.pHolder.pContent["soldier_"+s].gotoAndStop(a);
            trace("frame name is "+
      this.pHolder.pContent["soldier_"+s]._currentframe);
            trace("soldier_"+s+"'s width is "
+this.pHolder.pContent["soldier_"+s]._width + " and height is " +
this.pHolder.pContent["soldier_"+s]._height);
            trace("panel width is "+this._width+" and height is
"+this._height);
            var w:Number,h:Number,pw:Number,ph:Number,
sx:Number,sy:Number;
            // w, h = width and height of panel
            // pw, ph = width and height of person/soldier
            // sx, sy = the x,y to place the soldier ...have to
place before you scale...
            w = this._width;
            h = this._height;
            pw = this.pHolder.pContent["soldier_"+s]._width;
            pw = (shakinghands) ? 2*pw : pw;
            ph = this.pHolder.pContent["soldier_"+s]._height;
            trace(w + " " + h + " " + pw + " " + ph);
            var wr = w/(pw+20);
            var hr = h/(ph+20);
            var scaleRatio = 100;
```

```
            sx = ((w-10)/2) + 5;
            sy = h;
            if (shakinghands) {

      this.pHolder.pContent.attachMovie("person","p0", 475);
                  this.pHolder.pContent.p0.gotoAndStop(a);
                  var darken = new
Color(this.pHolder.pContent.p0);
                  darken.setRGB(0x000000);
                  this.pHolder.pContent.p0._alpha = 60;
            }
            if (pw > w) {
                  scaleRatio = 100*wr;
                  trace("soldier is wider than frame, scale by
100*wr");
                  if(ph>h) {
                        trace("soldier is also taller than
panel...");
                        if (t == "macro") {
                              scaleRatio = 100*Math.min(wr,hr);
                              trace("...and macro, so scale by
100*wr*hr instead");
                        } else if (ph*(scaleRatio) > h){
                              sy = (ph*(scaleRatio/100))+5;
                              trace("...and mono, so set sy");
                        }
                  }

            } else if (ph > h){
                  trace("soldier is not wider than panel, but is
taller than it");
                  if (t == "macro") {
                        trace("soldier is also macro, scale by
100*hr");
                        scaleRatio = 100*hr;
                  } else {
                        trace("soldier is also mono, set sy");
                        sy = (ph*(scaleRatio/100))+5;
                  }
            }
            this.pHolder.pContent["soldier_"+s]._x = sx;
            this.pHolder.pContent["soldier_"+s]._y = sy;
            trace("soldier is "+t+" and "+pw+"x"+ph+": moving to
"+sx+", "+sy+" and scaling by "+scaleRatio);
            if(shakinghands){
                  this.pHolder.pContent["soldier_"+s]._x = sx -
(scaleRatio/100)*pw/5;
```

```
                  this.pHolder.pContent.p0._x = sx +
(scaleRatio/100)*pw/5;
                  this.pHolder.pContent.p0._y = sy;
                  this.pHolder.pContent.p0._xscale = -scaleRatio;
                  this.pHolder.pContent.p0._yscale = scaleRatio;
                  trace("*************shaking
hands************");

      trace(this.pHolder.pContent.p0._x+","+this.pHolder.pContent
.p0._y+","+this.pHolder.pContent["soldier_"+s]._x);
            }
            this.pHolder.pContent["soldier_"+s]._xscale =
scaleRatio;
            this.pHolder.pContent["soldier_"+s]._yscale =
scaleRatio;
      }


      public function drawSoldier(activity:String) {

      }


      public function drawBackground(place:String,
zoomtype:Number):Void {

      }

//    public function createContent(s:Number, asset:String,
depth:Number, zoomtype:Number):Void {

//  }

      public function drawMe():Void {

            if (border_points[0] == null) {
                  trace("border points is null!  this should
never happen.");
                  return;
            }
            trace(" ====== drawing border ====== ");
            trace("setting line thickness");
            // --- draw border ---
            var _mask:MovieClip =
this.pHolder.pMask.createEmptyMovieClip("_mask", 1001);
            trace("creating new movieclip inside pMask");
            trace(this.pHolder.pMask._mask._name);
```

```
            // draw the rectangle in pMask._mask and set it as
the mask of pContent
            this.pHolder.pMask._mask.lineStyle(0,0xffffff,0);

            // this code is redundant, should break out into a
separate function
            this.pHolder.pMask._mask.beginFill(0x00ffff,50);

      this.pHolder.pMask._mask.moveTo(this.border_points[this.bor
der_points.length-
1].x,this.border_points[this.border_points.length-1].y);
            switch(this.border_shape){
            case "roundrect":
                  for (var i = 0; i < this.border_points.length-
2; i=i+3) {

      this.pHolder.pMask._mask.curveTo(this.border_points[i].x,
this.border_points[i].y, this.border_points[i+1].x,
this.border_points[i+1].y);

      this.pHolder.pMask._mask.lineTo(this.border_points[i+2].x,t
his.border_points[i+2].y);
                  }
                  break;
            default:
                  for( var i= 0; i<this.border_points.length;
i++) {

      trace("this.border_points["+i+"]="+this.border_points[i].x+
" "+this.border_points[i].y);

      this.pHolder.pMask._mask.lineTo(this.border_points[i].x,
this.border_points[i].y);
                  }
                  break;
            }
            this.pHolder.pMask._mask.endFill();
            trace("finished drawing the shape in _mask");

            // draw the unfilled shape (the border!) in pMask now
            this.pHolder.pMask.lineStyle(this.border_thickness,
this.border_color, 100);
            trace("moving to first point and about to draw...");
            // this code is redundant, should break out into a
separate function

      this.pHolder.pMask.moveTo(this.border_points[this.border_po
```

```
ints.length-1].x,this.border_points[this.border_points.length-
1].y);
            switch(this.border_shape){
                  case "roundrect":
                        for (var i = 0; i <
this.border_points.length-2; i=i+3) {

      this.pHolder.pMask.curveTo(this.border_points[i].x,
this.border_points[i].y, this.border_points[i+1].x,
this.border_points[i+1].y);

      this.pHolder.pMask.lineTo(this.border_points[i+2].x,this.bo
rder_points[i+2].y);
                        }
                        break;
                  case "bleed":
                  case "circlebleed":
                        break;
                  default:
                        for( var i= 0;
i<this.border_points.length; i++) {

      trace("this.border_points["+i+"]="+this.border_points[i].x+
" "+this.border_points[i].y);

      this.pHolder.pMask.lineTo(this.border_points[i].x,
this.border_points[i].y);
                        }
                        break;
            }
            trace("finished drawing the shape of the border");

            trace("now set this movieClip to be the Mask of the
pContents movieClip ");

      this.pHolder.pContent.setMask(this.pHolder.pMask._mask);
      }


}
```

Comic.as

The Comic class contains an Array of all of the Panels. It has its own x,y (for the location of the comics grid in the UI, see following section), and controls the size of the grid squares.

```
class Comic {
```

```
	var _x:Number, _y:Number;
	var _height:Number, _width:Number;
	var blocks_wide:Number, blocks_high:Number;
	var blocksize:Number;
	var panelList:Array;

	var comicsHolder:MovieClip;

	public function Comic() {
		this._x = 10;
		this._y = 64;
		this._width = 600;
		this._height = 360;
		this.blocks_wide = 10;
		this.blocks_high = 6;
		this.blocksize = 60;
		this.panelList = new Array();

		this.comicsHolder =
_root.createEmptyMovieClip("comicsHolder", 10);
		this.comicsHolder._x = this._x;
		this.comicsHolder._y = this._y;
		var cCanvas =
comicsHolder.createEmptyMovieClip("cCanvas", 100);
		var cMask =
comicsHolder.createEmptyMovieClip("cMask", 200);
		this.createMask();
	}

	private function createMask():Void {
		with(this.comicsHolder.cMask){
			beginFill(0x000000,50);
			moveTo(0,0);
			lineTo(0,this._height);
			lineTo(this._width, this._height);
			lineTo(this._width, 0);
			lineTo(0,0);
			endFill();
		}
		trace(this.comicsHolder.cCanvas._name);

	this.comicsHolder.cCanvas.setMask(this.comicsHolder.cMask);
	}

	public function
setDimensions(w:Number,h:Number,bw:Number,bh:Number):Void {
		this.blocksize = Math.min(Math.floor(w/bw),
Math.floor(h/bh));
```

```
            this._width = this.blocksize * bw;
            this._height = this.blocksize * bh;
            this.blocks_wide = bw;
            this.blocks_high = bh;
      }

      public function addPanel( p:Panel):Void {
            this.panelList.push(p);
      }


      public function getPanel( index:Number):Panel {
            return this.panelList[index];
      }

      // change out pw and ph with st:String, sn:Number
      public function setPanelBorderDimensions( index:Number,
bs:String, bt:Number, bc:Color, bw:Number, bh:Number):Void {
            this.panelList[index].setBorder(bs,bt,bc,"solid");
            this.panelList[index].setDimensions(bw,bh);
            // switch statement here for st/sn
      }

      // replace this function with something more meaningful?
      public function createPanelContent( index:Number,
pa:String, pt:String):Void {
            this.panelList[index].createContent(index,pa,pt);
      }

      public function drawPanel( index:Number ):Void {
            this.panelList[index].drawMe();
      }

      public function drawAllPanels():Void {
            for (var i = 0; i<this.panelList.length; i++) {
                  this.panelList[i].drawMe();
            }
      }

}
```

p4.fla

This is the ActionScript in the **actions** layer on the main timeline. XML parsing and all of the other code orchestration (instantiation of the Comic and Panel objects) occurs here.

```
// code to change the color of a movie clip...
```

```
// var melanin:Color = new Color(person1.p);
// melanin.setRGB(0x000000);

var thisComic = new Comic();

initUI();

// xml code example from what i was working on with paul
thisXML = new XML();
thisXML.ignoreWhite = true;
thisXML.onLoad = myLoad;
var urlstring:String = "comic.xml";
var panels:Array = new Array();
var currentX:Number = 0;
var currentY:Number = 0;
var highestY:Number = 0;
trace("currentX is "+currentX);
var fullyLoaded:Boolean = false;

var myHeight:Number = Stage.height;

thisXML.load(urlstring);

function myLoad () {
	// this is also quite truly a hack and will have to be
fixed first thing
	trace(thisXML);
	currentX = 0;
	xml_comic = thisXML.firstChild;
	xpd = xml_comic.childNodes[4];
	var dp = {w: parseInt(xpd.firstChild.firstChild.nodeValue),
h: parseInt(xpd.childNodes[1].firstChild.nodeValue), gb:
parseInt(xpd.childNodes[2].firstChild.nodeValue), ga:
parseInt(xpd.childNodes[3].firstChild.nodeValue), bs:
xpd.childNodes[4].childNodes[0].firstChild.nodeValue, bt:
parseInt(xpd.childNodes[4].childNodes[1].firstChild.nodeValue),
bc: xpd.childNodes[4].childNodes[2].firstChild.nodeValue, t:
xpd.childNodes[5].firstChild.nodeValue};
	trace("defaults:: width: "+dp.w + "; height: "+dp.h+";
gutter:"+dp.gb+" "+dp.ga+"; border: "+dp.bs+" "+dp.bt+"
"+dp.bc+"; type: "+dp.t);
	trace("there are
"+xml_comic.childNodes[5].childNodes.length+" panels");
	all_panels = xml_comic.childNodes[5].childNodes;
	var nodey:XMLNode;
	for (i:Number = 0; i < all_panels.length; i++){
		var temp = {w: dp.w, h: dp.h, a: "standing", t: dp.t,
gb: dp.gb, ga: dp.ga, bs: dp.bs, bt: dp.bt, bc: dp.bc};
```

```
            for (j:Number = 0; j<
all_panels[i].childNodes.length; j++){
                  nodey = all_panels[i].childNodes[j];
                  switch(nodey.nodeName){
                        case "panel-desc":
                              if(nodey.firstChild.nodeName
=="action") {

      trace(nodey.firstChild.firstChild.firstChild.nodeValue);
                                    temp.a =
nodey.firstChild.firstChild.firstChild.nodeValue;
                              }
                              if(nodey.childNodes[1].nodeName
=="composition") {
                                    temp.t =
nodey.childNodes[1].firstChild.firstChild.nodeValue;
                                    trace(temp.t);
                              }

                        break;
                        case "size":
                              trace("the case was size"+i);

      switch(nodey.firstChild.firstChild.nodeValue){
                                    case "big":
                                          trace("panel "+i+" is
big");
                                          temp.w = 2*dp.w;
                                          temp.h = 2*dp.h;
                                    break;
                                    case "small":
                                          temp.w = 0.5*dp.w;
                                          temp.h = 0.5*dp.h;
                                    break;
                                    case "wide":
                                          temp.w=3*dp.w;
                                          temp.h=dp.h;
                                    break;
                                    case "normal":
                                    default:
                                          temp.w = dp.w;
                                          temp.h = dp.h;
                                    break;
                              }

                        break;
                        case "gutter":
                              trace ("gutter ahoy!");
```

```
                        break;
                        case "border":
                              trace ("new border spec!!!! ****");
                              //this will foul up if only one
childnode of border is here, e.g.
                              temp.bs =
nodey.firstChild.firstChild.nodeValue;
                              temp.bt =
parseInt(nodey.childNodes[1].firstChild.nodeValue);
                              temp.bc =
nodey.childNodes[2].firstChild.nodeValue;
                        break;
                  }
                  trace (nodey.nodeName + "has type " +
nodey.nodeType);
            }

      trace(temp.w+","+temp.h+","+temp.a+","+temp.t+","+temp.bs+"
,"+temp.bt+","+temp.bc);
            currentX +=dp.gb;
            if (currentX+temp.w > 600){
                  currentX=0;
                  currentY=currentY+highestY+40;
            } else {
                  highestY = (temp.h > highestY) ? temp.h :
highestY;
            }

            thisComic.addPanel(new
Panel(currentX+dp.gb,currentY,i+1,i+1,'comicsHolder',
thisComic.blocksize));
            trace("currentX is "+currentX);
            currentX += temp.w;
            thisComic.setPanelBorderDimensions(i,temp.bs,
temp.bt, temp.bc, temp.w, temp.h);
            thisComic.createPanelContent(i,temp.a,temp.t);
            thisComic.drawPanel(i);
      }

trace("the whole length is actually: "+currentX);
fullyLoaded = true;
}

// Must scale by +/- 5% to get rid of artifacts ...but I don't
care now.
// _root._xscale = 105;
// _root._yscale = 105;
```

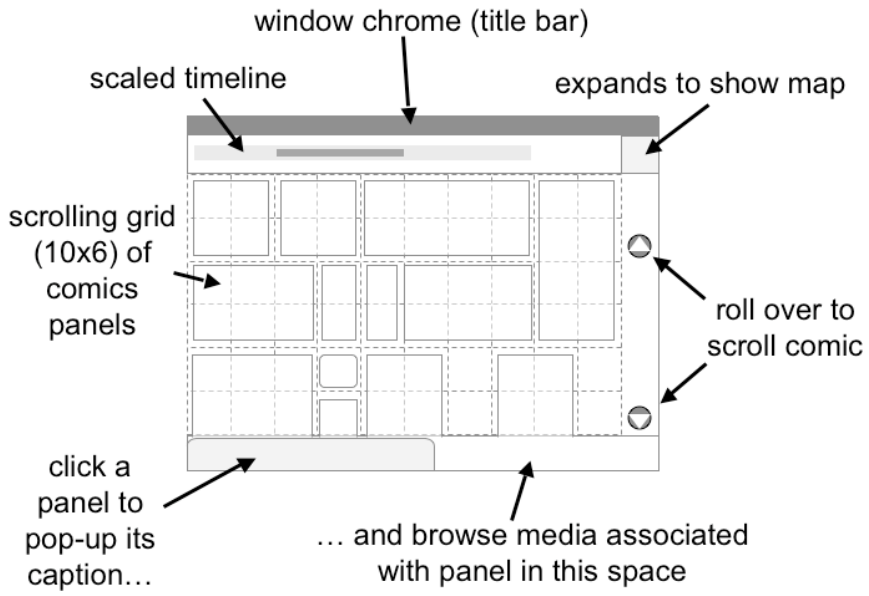
```
// Scrolling code ...could it go in tempframe?  this is probably
an egregious kludge...
comicsHolder.cCanvas.onEnterFrame = function() {
      if(fullyLoaded) {
            comicsHolder.cCanvas._y += 20*((myHeight/2)-
(_ymouse+comicsHolder.cCanvas._y))/myHeight;
            comicsHolder.cCanvas._y =
constrain(comicsHolder.cCanvas._y, 0, -currentX);
      }
}
//aha!  it could!

function constrain(num:Number, n2:Number, n3:Number):Number {
      var minx = Math.min(n2, n3);
      var maxx = Math.max(n2, n3);
      return Math.min(Math.max(minx,num),maxx);
}

function initUI() {
      // create top ui elements
      attachMovie("dark_bar", "topbar", 15);
      topbar._x = 0;
      topbar._y = 0;
      topbar._width = 640;
      topbar._height= 64;
      attachMovie("map_mc", "map", 30);
      map._x = 640;
      map._y = 0;

      // create bottom ui elements
      attachMovie("dark_bar", "bottombar", 20);
      bottombar._x = 0;
      bottombar._y = 424;
      bottombar._width = 640;
      bottombar._height= 56;
      var captionbar:MovieClip =
_root.createEmptyMovieClip("captionbar",1000);
      captionbar._x = 0;
      captionbar._y = 424;
      captionbar.createEmptyMovieClip("main",100);
      captionbar.createEmptyMovieClip("cmask",200);
      captionbar.main.attachMovie("yellow_bar", "caption_bg1",
10);
      captionbar.main.attachMovie("yellow_bar", "caption_bg2",
15);
      captionbar.main.createTextField("the_caption",20, 7, 7,
captionbar.main.caption_bg1._width-14,
captionbar.main.caption_bg1._height-7);
```

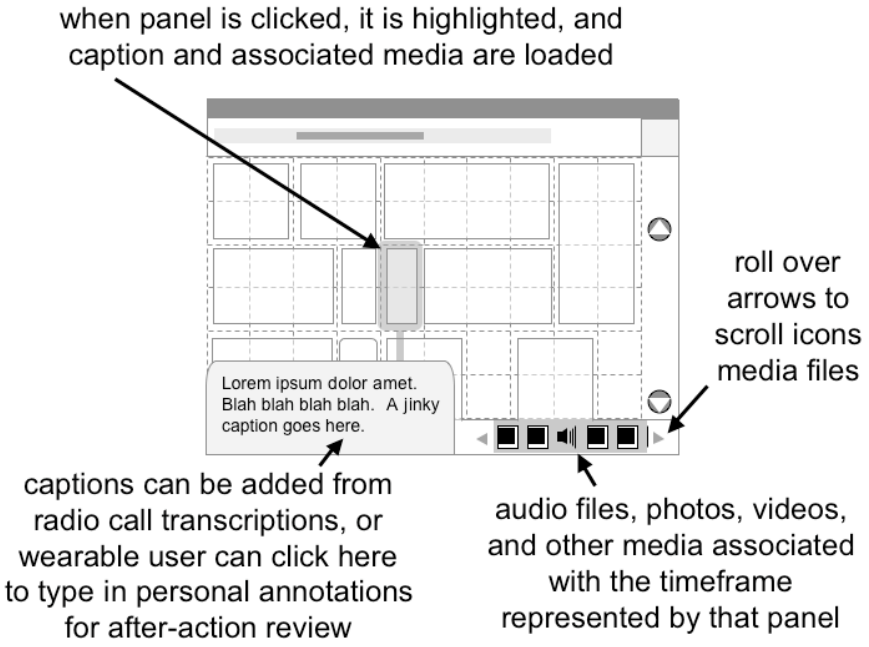
```
	with(captionbar.main.the_caption){
		multiline = true;
		wordWrap = true;
		_visible = true;
		text = "(Click on a panel to see or enter caption
data for it.)";
		var captionFormat:TextFormat = new TextFormat();
		captionFormat.font = "Verdana";
		captionFormat.size = 16;
		setTextFormat(captionFormat);
	}
	with(captionbar.cmask){
		moveTo(0,0);
		beginFill(0x000000, 50);
		lineTo(0, captionbar.main.caption_bg1._height);
		lineTo(captionbar.main.caption_bg1._width,
captionbar.main.caption_bg1._height);
		lineTo(captionbar.main.caption_bg1._width, 0);
		lineTo(0,0);
		endFill();
	}
	captionbar.cmask._y=-102;
	captionbar.main.caption_bg1._x = 0;
	captionbar.main.caption_bg1._y = 0;
	//captionbar.caption_bg1._width = 320;
	captionbar.main.caption_bg2._x = 5;
	captionbar.main.caption_bg2._y = 3;
	captionbar.main.caption_bg2._width =
captionbar.main.caption_bg1._width - 10;
	captionbar.main.caption_bg2._height =
captionbar.main.caption_bg1._height - 3;
	captionbar.main.setMask(captionbar.cmask);
}
```

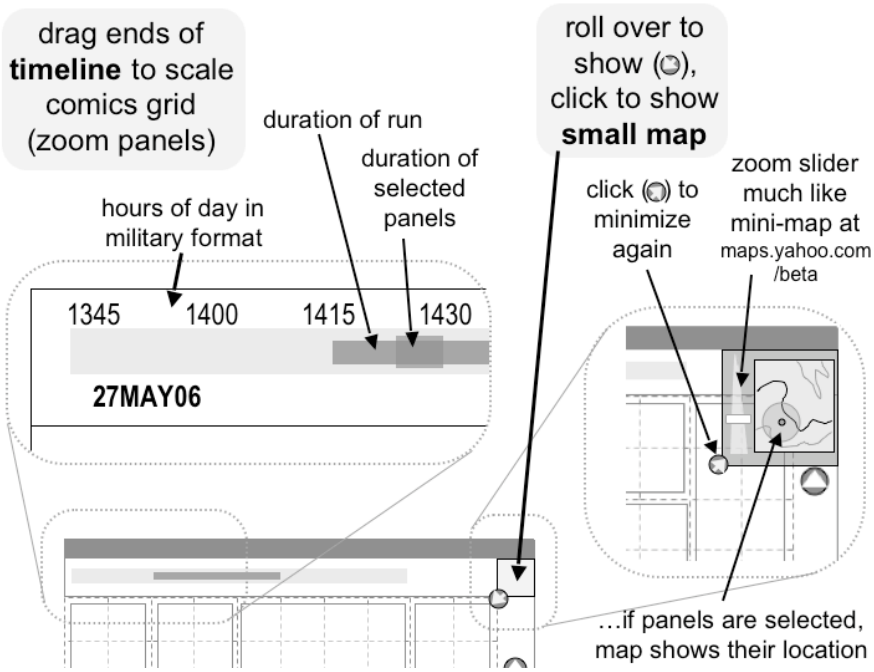
Appendix 8: User Interface Wireframes

Suggested UI for browsing comics summaries. Resizes to fit any 4:3 ratio (640x480, 800x600, 1024x768, etc.) screen.

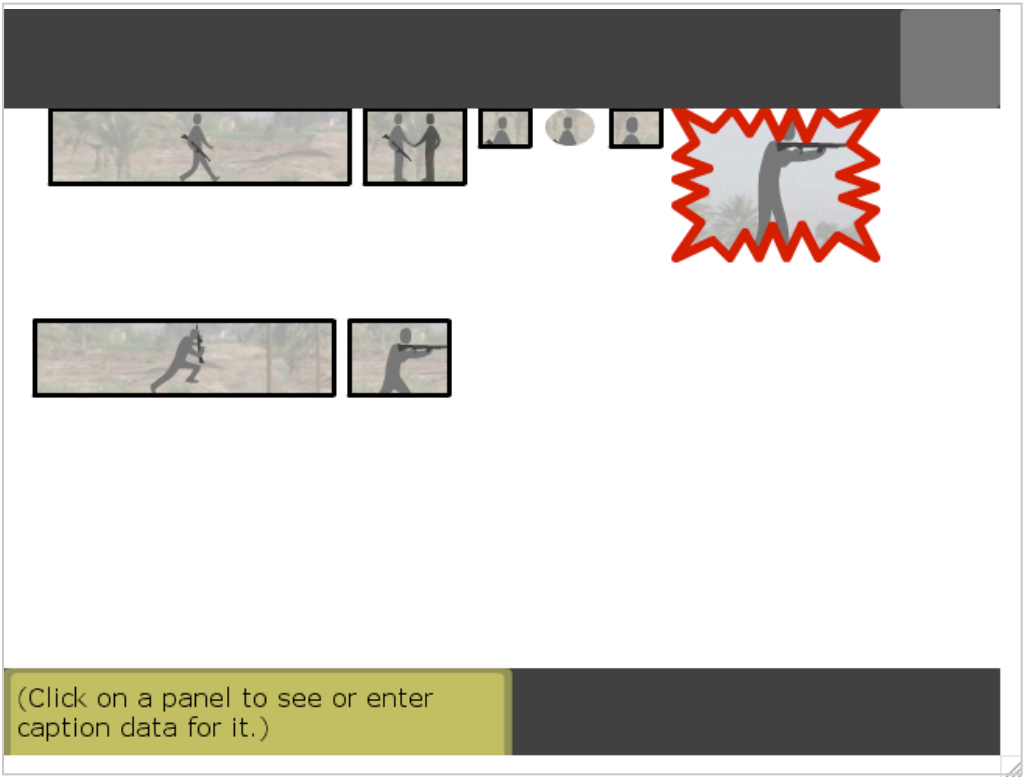
For a 4:3 ratio screen:







Screenshot from the Flash prototype from the code in *Appendix 7*:



(As one can see, this is a very very rough UI: for starters, there is no panel packing, all panels have same background, and most of UI is just stubbed in.)