

Ishan Anand  
MIT Department of Electrical Engineering and Computer Science  
Undergraduate Advanced Project  
May 18, 2006

Advisors:  
Drew Endy,  
Assistant Professor, Biological Engineering

Sriram Kosuri  
Ph.D. Candidate, Biological Engineering

# GeneJax: A Prototype CAD tool in support of Genome Refactoring

**Abstract:** Refactoring is a technique used by computer scientists for improving program design. The Endy Laboratory has adapted this process to make the genomes of biological organisms more amenable to human understanding and design goals. To assist in this endeavor, we implemented GeneJax, a prototype JavaScript web application for the dissection and visualization stages of the genome refactoring process. This paper reviews key genome refactoring concepts and then discusses the features, development history, user-interface, and underlying implementation issues faced during the making of GeneJax. In addition, we provide recommendations for future GeneJax development. This paper may be of interest to engineers of CAD tools for synthetic biology.

---

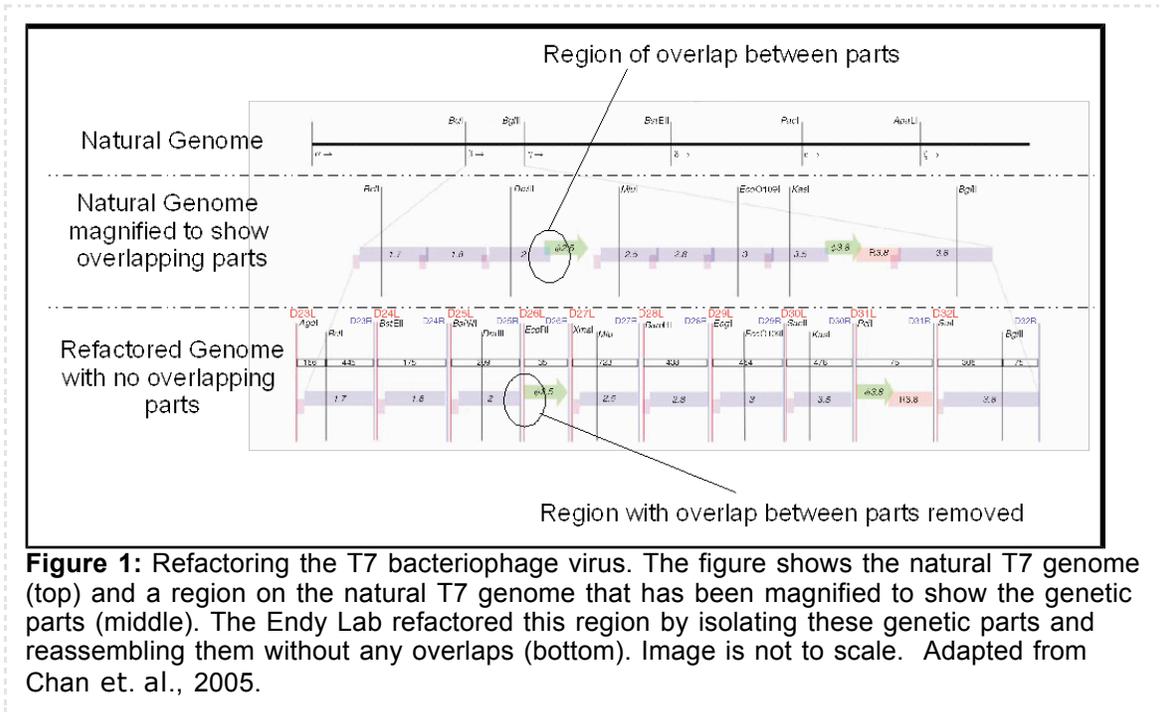
## Introduction: Genome Refactoring

Refactoring is a technique traditionally used by software engineers to redesign computer software (Fowler et. al., 1999). By refactoring, engineers modify the design of an existing program without adding new features or functionality. Instead the new design improves program readability and maintainability:

Refactoring does not fix bugs or add new functionality. Rather it is designed to improve the understandability of the code or change its structure and design, and remove dead code, to make it easier for human maintenance in the future. In particular, adding new behavior to a program might be difficult with the program's given structure, so a developer might refactor it first to make it easy, and then add the new behavior. (Wikipedia contributors, 2006)

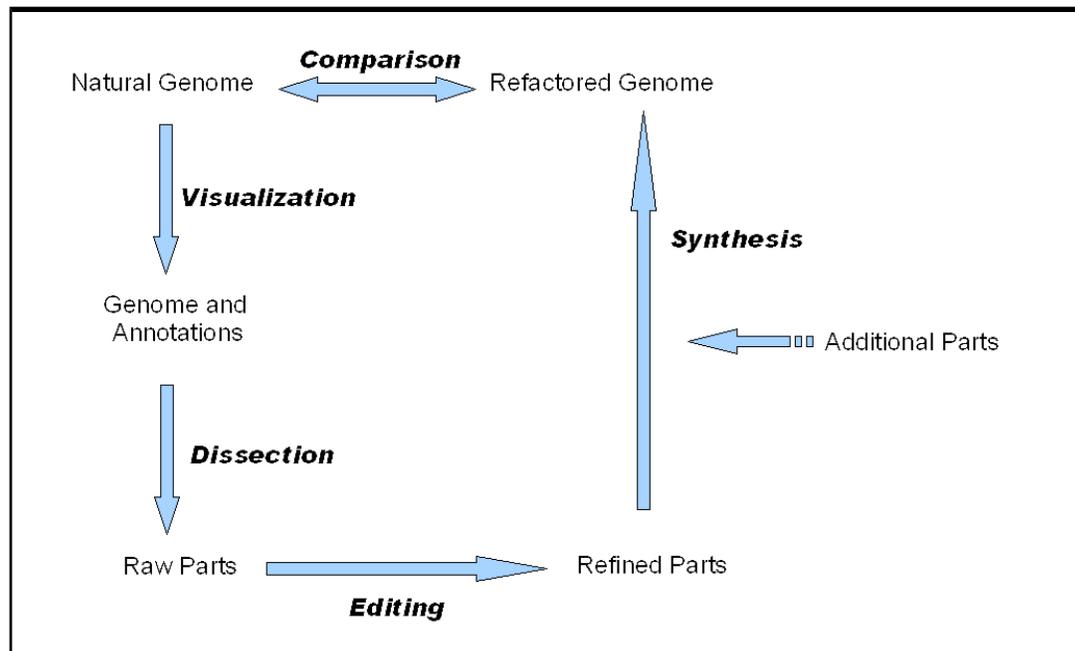
Inspired by this technique, the MIT Endy Lab has begun refactoring genomes. In particular, the team seeks to precisely understand how the encoded genetic elements interact with one another to generate an entire functioning organism. However, the sequences encoded by natural genomes are usually hard to study for a variety of reasons. For example, in natural genomes multiple genes and functions often overlap on the same DNA sequence making it hard to determine each gene's contribution to the organism. One way to simplify such a genome would be to engineer a new genome where each region of the DNA sequence encodes a single genetic function. By isolating these functional elements from one another, each element's function becomes easier to model and simpler to manipulate. As a result, this "refactored" genome is easier for humans to understand and modify.

For example, the Endy Lab recently disassembled the genome of bacteriophage T7, a bacterial virus, and constructed a new man-made "T7.1" virus sequence (Chan et. al., 2005). This T7.1 virus was functionally similar to the natural T7 virus but with a more structured design that removed overlaps between genetic elements. Removing these overlaps allowed the team to independently manipulate each specific gene element, and by doing so, fully describe its function. Figure 1 compares a region from the T7 virus and its refactored T7.1 variant, illustrating the improved isolation between genetic elements in the refactored version.



**Figure 1:** Refactoring the T7 bacteriophage virus. The figure shows the natural T7 genome (top) and a region on the natural T7 genome that has been magnified to show the genetic parts (middle). The Endy Lab refactored this region by isolating these genetic parts and reassembling them without any overlaps (bottom). Image is not to scale. Adapted from Chan et. al., 2005.

The process of genome refactoring can be broken down into five steps (visualization, dissection, editing, synthesis, comparison) outlined in Figure 2. First a naturally occurring genome and its annotations are examined during the *visualization* phase. These annotations indicate the known functionality of various sections in the genome DNA sequence. Using this knowledge the genome is *dissected* into parts that isolate each gene function from the natural genome. Since naturally occurring sequences often have overlapping functional elements, the resulting parts may overlap or contain sequences for additional, unintended functions. These naturally occurring parts must therefore be *edited* to refine their isolation from other genetic elements. The refined parts are then *synthesized* into a refactored genome. Additional parts may come from other refactored genomes as shown in the figure. Finally, the naturally occurring and refactored genome may be *compared* against each other to examine the differences where the genome has and has not been changed. The refactoring process described here is adapted from the genome design algorithm presented in the supplementary materials for (Chan et. al., 2005).



**Figure 2:** The Genome Refactoring process. First, a natural genome is visualized and then dissected into raw parts. Next, these raw parts are edited to refine their function. The refined parts are then combined with parts from other genomes to synthesize a new refactored genome. Finally, the refactored genome and natural genome are compared to examine where the genome has and has not been changed. Adapted from Chan et. al., 2005.

The Endy Lab has now begun work on a “T7.2” virus and is looking for software tools to assist in this process. Instead of rearranging existing genetic elements within the organism (as was done with T7.1), the T7.2 design will actively remove gene elements and substitute other elements with new genes from other genomes:

Moving beyond our design of T7.1, we will actively erase or delete elements of unknown function. In addition, efforts will be made to remove unknown genetic elements... To attempt to make our modeling of gene expression easier, we will use standard synthetic elements in place of the natural elements that regulate transcription and translation. (T7.2, 2005)

The original T7.1 virus was designed using existing tools such as Vector NTI and custom perl scripts coerced into ad hoc refactoring tasks. In particular, these tools do not easily allow the arbitrary definition genetic parts and their reassembly into multiple refactored genome variants. As the goals of T7.2 increase the complexity of gene edits beyond that of T7.1, the team seeks a new tool to streamline this process of part definition and manipulation.

Software programs for *almost* all phases of the refactoring process already exist. For the *visualization* phase there are a plethora of tools available as stand alone programs or on the web. For example, applications such as VectorNTI from Invitrogen and Workbench from CLC Bio offer excellent visualization features. In addition, the National Center for BioInformatics, the University of California at Santa Cruz, and University of California at Berkeley all maintain online genome browsers for visualizing genomes and their annotations. Online tools for the *editing* and *synthesis* phase are already under development at the MIT Registry of Standard Biological Parts. The Registry supports the *synthesis* of genes from the parts in its database

using BioBricks and more recently as simple blunt-end concatenation (Sriram Kosuri, personal communication). The Registry is also developing a JavaScript program for *editing* part sequences from within the web browser (Randy Rettberg, personal communication). Finally, BioViz is a web application that uses SVG and JavaScript for *comparing* genomes (Lewis, n.d.). This list is by no means exhaustive, but we are unaware of any tools designed specifically for the *dissection* phase of the refactoring process.

This lack of tools for the dissection stage led us to the GeneJax project. Specifically, GeneJax was intended to let users define and extract arbitrary parts from a natural genome sequence. This dissection functionality also required a method to visualize newly defined parts and existing annotations within a genome. As a result GeneJax necessarily had to minimally implement visualization features already handled by the tools described previously.

The development of GeneJax used Google Maps as the starting point for these visualization features. We were influenced by Google Maps for two reasons. First, we reasoned that at an abstract level, the functionality of Google Maps could be applied to the visualization and navigation of genomic sequences. Both genomes and geographic mapping programs must provide a localized view of a very large data set at multiple resolutions. Put more plainly, the map of the human genome is a *map* and Google Maps is an excellent tool for traversing maps. These similarities are summarized in the following table.

<b>Feature</b>	<b>Google Maps</b>	<b>GeneJax</b>
Displays relevant subset of larger a database	Maps a point of interest from a map of the entire world.	Maps gene sequence of interest from an entire genome.
Click-drag	Mouse "click-and-drag" movement allows users to scan adjacent maps without reloading the browser page.	Mouse "click-and-drag" movement allows users to scan adjacent genes or gene base-pairs without reloading the browser page.
Zoom In/Out	Users can zoom in to street level details, or zoom out to a world map view.	Users can zoom in to see an individual gene or base pairs and zoom out to view the whole genome.
Overlaid maps	Satellite views, road map views, and an overlay	Functional, sequence, and overlay views

Second, Google Maps and the accompanying AJAX movement demonstrated that application-like interactivity can be built into a web based tool. This would allow us to go beyond the pure visualization capabilities of Google Maps and add application-like functionality for defining and manipulating genetic parts. Thus a Google-like map for the genomes was to provide the navigational foundation upon which we would build the features for defining parts from the gene sequence.

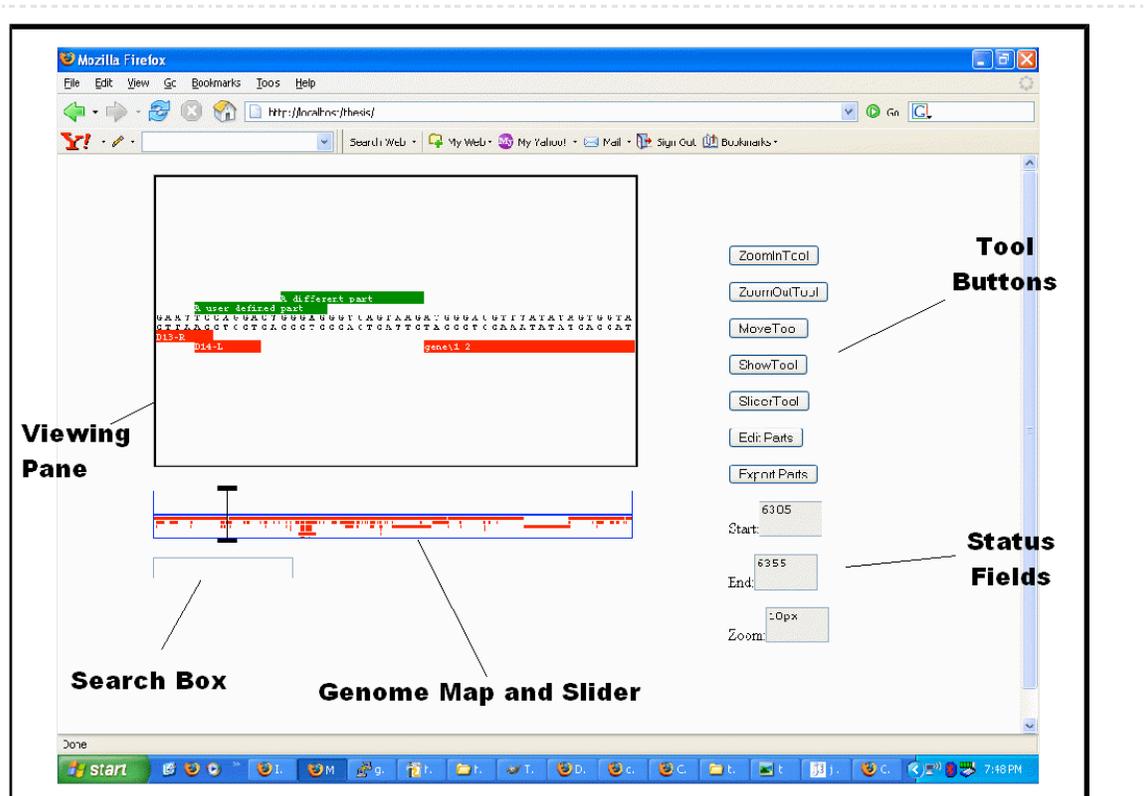
In the following sections we will describe the details of the GenJax project and the lessons learned. We first describe the GeneJax features that have been completed and their development from an end-user perspective. Next we don our programming hat and explore key implementation issues faced during the project. Finally, we speculate and propose enhancements for future versions of the project.

---

## GeneJax Features

We now describe the current functionality of the GeneJax program, and the strengths and weaknesses of those features from the perspective of the end-user. This section is not intended to be a user manual for the program. Instead we critically assess the history, quality and usefulness of the implemented features, since that information will be the most useful to designers of future genome refactoring tools. Our discussion starts with GeneJax's layout, and then moves on the basic map visualization and navigation features. These features form the foundation for the part definition functions that we explore at the end of this section.

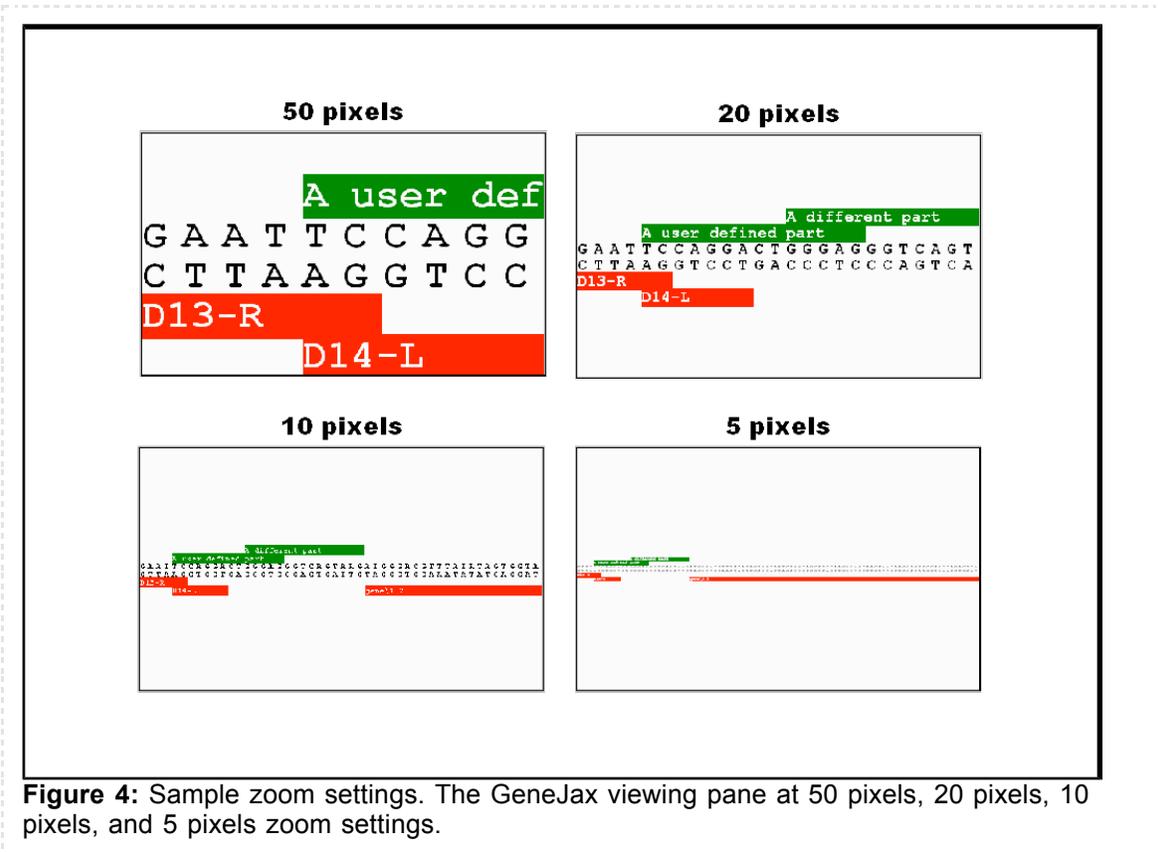
Before discussing the strengths and weaknesses of GeneJax, it is helpful for the reader to get a feel for the program's interface. A screen shot of GeneJax running in the Firefox browser is shown in Figure 3. In the top left a large box forms the *viewing pane* that displays the gene sequence, annotations, and parts. Beneath the viewing pane, is a *map* of the entire genome sequence and a *slider* indicating the current position of the viewing pane on the map. Beneath the map is a *search* field that lets the user search for a region of interest. The lower right contains *status fields* that indicate the absolute base pair position of the sequence in the viewing pane and the current zoom setting. Finally the upper right contains *tool buttons* that enable the features of the program.



**Figure 3:** The GeneJax User Interface. A screen shot of the GeneJax program, identifying the key features of the program's layout. The program has a *viewing pane* that displays genome data; a *search box*, *genome map*, and *status fields* for navigating the genome; and *tool buttons* for enabling the different functions of the program.

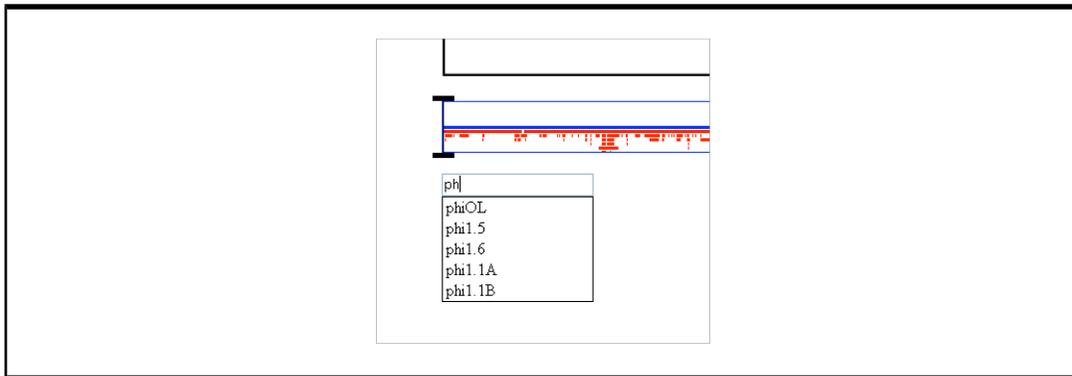
Navigation in GeneJax is similar to the basic map navigation features of Google Maps. The viewing pane shows the base pair sequence, annotations, and parts for a region of interest on the genome. By using the MoveTool, users can click-drag to examine nearby regions of the genome. In the current implementation mouse movements in the vertical direction are ignored

during the click-drag process, so only horizontal movements are translated into viewing pane navigation. Users can also zoom in and out. Zoom levels are indicated in the status field and indicate the font size of the text displayed in the viewing pane. Some examples of the viewing pane at different settings are shown in Figure 4. The available zoom levels range from 250 pixels (showing two base pairs on the screen) to a single pixel (showing 500 base pairs on the screen) but not all of the settings in this range are practically useful. Particularly problematic, text is completely unreadable for small zoom settings that would be needed to completely display large sized gene features (i.e. in excess of a 50 to a 100 base pairs long). For all intents and purposes, the maximum usable zoom range is from 50 pixels (10 base pairs per screen) to 10 pixels (50 base pairs per screen). Even combined with the other navigational aides described below, this is barely usable for dissecting large sized parts.



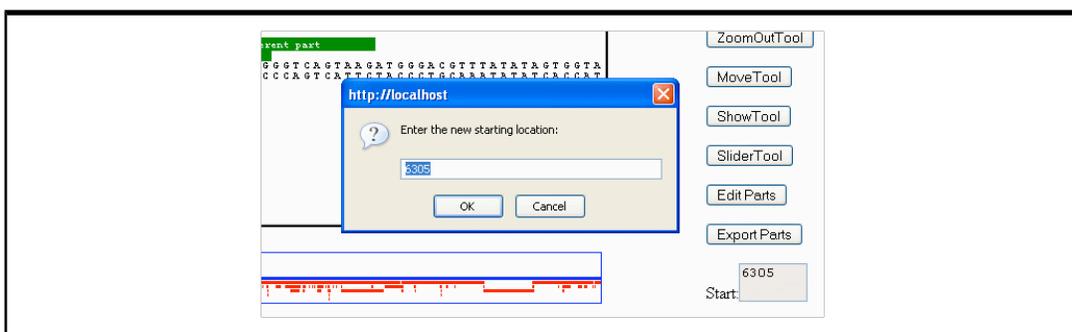
The preceding Google Maps navigation features were the only ones originally planned for GeneJax. However it quickly became clear that moving about a genome with click-drag can be cumbersome and additional navigational aides were needed. The most obvious problem is that trying to move across thousands of base pairs can mean click-dragging thousands of times. Our early assumption was that users could solve this the same way they do with Google Maps: by zooming out, click-dragging, and then zooming back in. This was wrong for three reasons.

First, as mentioned above, our zoom implementation was unreadable at scales showing more than a 100 base pairs. This made seeing the whole genome (more than 40,000 base pairs) and then zooming in on a point of interest nearly impossible. Repairing the zoom function was a technological hurdle we did not have time to correct. (The reasons for this hurdle will be described later in the technology section).

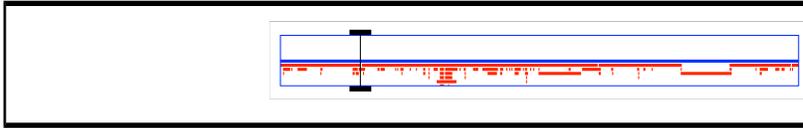


**Figure 5:** Search box with auto suggest. GeneJax automatically suggests possible queries (i.e. matching annotations) as the user types "ph" into the search box.

Second, we failed to realize that search is a crucial part of Google Maps. Users do not start their mapping sessions with a map but with a search query that retrieves a location of interest. This is even more important when mapping a genome, since the user is more likely to jump around from annotation to annotation along the genome than someone gradually traversing a direction of travel in a geographic map like Google Maps. We solved this problem by adding a search box to GeneJax. By typing a part name or an annotation into this search field, the viewing pane will automatically jump to the region of interest. In addition, since all the possible points of interest are known ahead of time (i.e. an annotation or a newly defined part), the search box will suggest possible search queries as the user types into the search box. This is shown in Figure 5, where the program has suggested possible queries as the user types "ph" into the search box. The auto suggest implementation was adapted from (Zakas, et. al. 2006). We also let users jump to an arbitrary base-pair location by clicking on the start position indicator as shown in Figure 6. As obvious as it may seem in retrospect to add a search box, it was not an anticipated feature nor the first solution that occurred to us for this problem.

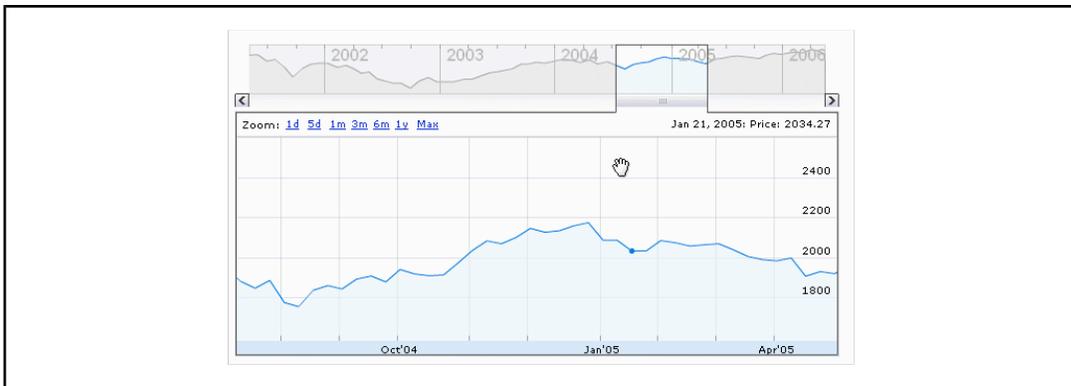


**Figure 6:** Jumping to a base pair location. A user moves the viewing pane by entering a base pair position in the genome.

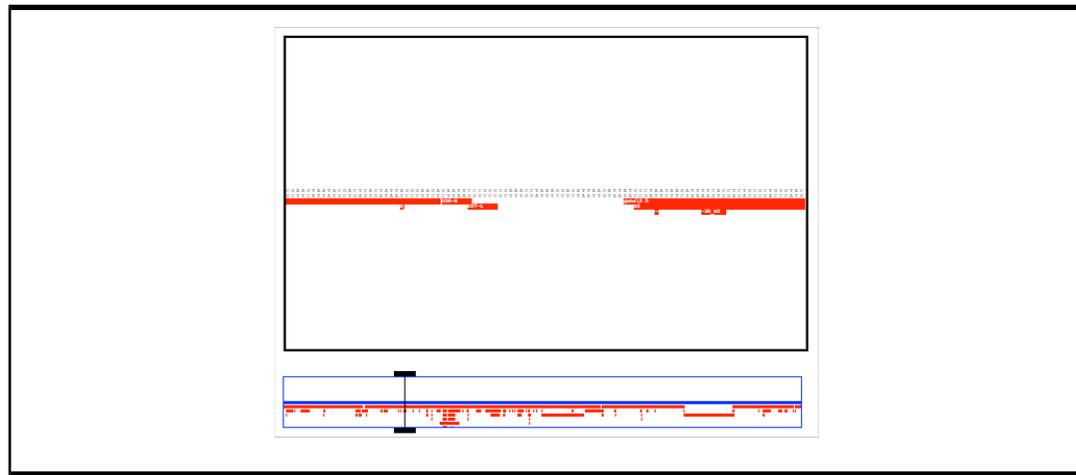


**Figure 7:** Genome map and slider. The slider indicates the viewing pane's current position on a map of the entire genome. The viewing pane can also be moved by click-dragging the slider or clicking on the map.

Third, at scales showing individual parts or base pairs, users felt lost and requested a visual indication of their position in the whole genome. We solved this by providing a graphical map of the entire genome and a slider indicating the current position, as shown in Figure 7. As the user click-drags to new locations in the viewing pane the slider's position on the map is dynamically updated. In addition, users can move the viewing pane to a new location by dragging the slider or clicking a position on the map. The success of the slider is somewhat questionable. We had initially planned that the slider's width should indicate the area under inspection, similar to Google Finance as shown in Figure 8. Unfortunately, given the length of the genome and the width of the map, a single pixel on the map corresponds to about 80 base pairs (41,326 base pairs in the genome divided by 500 pixels map width). Because the viewing pane is unreadable at this scale, all useful zoom levels are essentially condensed to the minimum width slider, a single pixel. This high base pair to pixel ratio also makes the graphical features on the map less indicative of the actual structure of the genome than they should be. For example, in Figure 9 the slider map indicates we should expect to see two continuous annotations in the viewing pane, but instead we see something quite different. In later sections we will explore some solutions to this problem but in the meantime note that by contrast Google Finance displays financial data for only the past five years, which would correspond to a genome of less than 2000 base pairs (5 years times 365 days). This much more usable "base pairs" to pixels ratio of 3.65 makes their slider possible. Last but not least it should be noted that users can move to any absolute position in the genome sequence (i.e. "take me to base pair 10,431"), which is essentially a different kind of search.

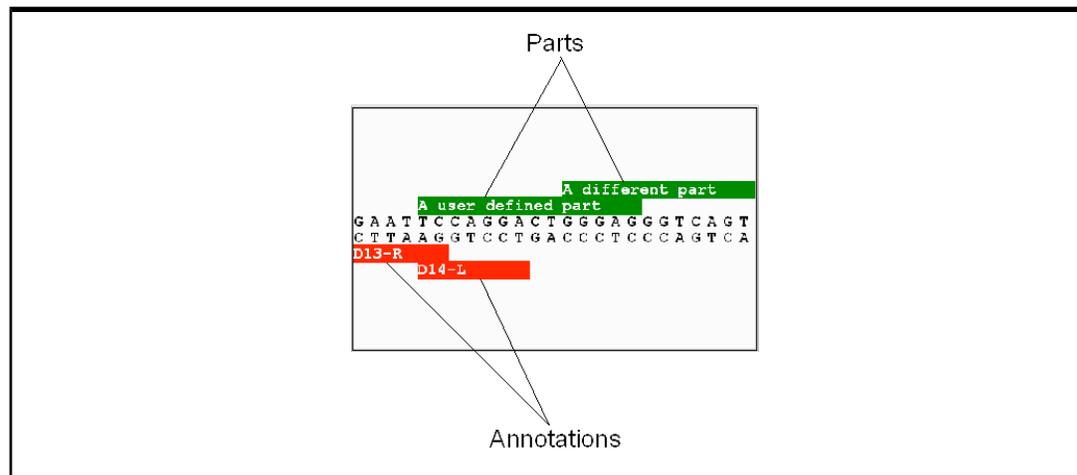


**Figure 8:** The Google Finance map and slider. The width of the slider (top) indicates the area under examination in the viewing pane (bottom).



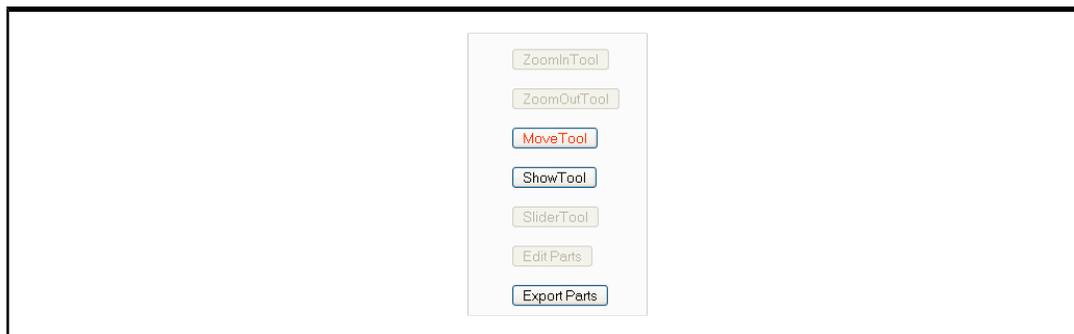
**Figure 9:** Genome map inaccuracies. A screen shot illustrating how the features on the genome map do not always accurately indicate the actual layout of the genome. In the figure, the viewing pane shows a clear gap in the annotations. However, in the genome map (the region under the slider) this complex layout is reduced down to three continuous annotations. (Only two of these annotations are visible in the figure because the third one is only a single pixel wide and obscured by the slider itself.) These inaccuracies are caused by the high ratio between the length of the genome (in base pairs) and the length of the map (in pixels).

GeneJax's *raison d'être* is to support the dissection of a genome into arbitrary parts. Traditionally genome visualizers place annotations located on the main DNA strand above the gene sequence and annotations associated with the complement strand below the gene sequence. For refactoring the distinction between parts and annotations is more important than knowing if a part or annotation is located on main strand or the complement strand. Thus annotations are separated from parts by displaying annotations below the sequence and by displaying parts above the main sequence as shown in Figure 10. Users create a part by clicking on a base pair to define the part's starting point and then control-clicking on a base pair to define its ending point. Subsequent control-clicks can resize the part by moving its end point. With this dissection paradigm, very large parts can be created by defining a starting point and then navigating (e.g via search or click click-drag) to a different position to set the part's end point. Users can also select their own names for parts. These part names are valid search queries and will show up in the search box autosuggestion field when appropriate. Finally users can export all defined parts to FASTA format using the ExportTool.



**Figure 10:** Annotations and parts. The GeneJax viewing pane displays parts above the genome sequence and annotations below it, irrespective of the annotations' association with the main or complement strand of the sequence.

Last but not least, from a usability standpoint the tool buttons were well-intentioned but badly conceived as toggles. Clicking on a tool button enables its functionality but also disables all other tool buttons, as shown in Figure 11 for an active MoveTool button. The purpose of this work flow was to disambiguate the purpose of certain mouse actions. For example, depending on the currently active tool, clicking on a region in the viewing pane could start the click-drag movement process or define the starting point for a new part. Unfortunately, the toggle paradigm forced users to first disable the current tool and then enable the new one when switching between tools. This turned out to be a cumbersome work flow and we painfully watched users make repeatedly mix up this process (although they often recovered quickly). In addition, the usage paradigm was inconsistent as some tools were not toggles (e.g. ExportTool), did not appear to be toggles (e.g. ZoomInTool) and toggling excluded features that were not necessarily mutually exclusive (e.g. click-drag movement and searching).

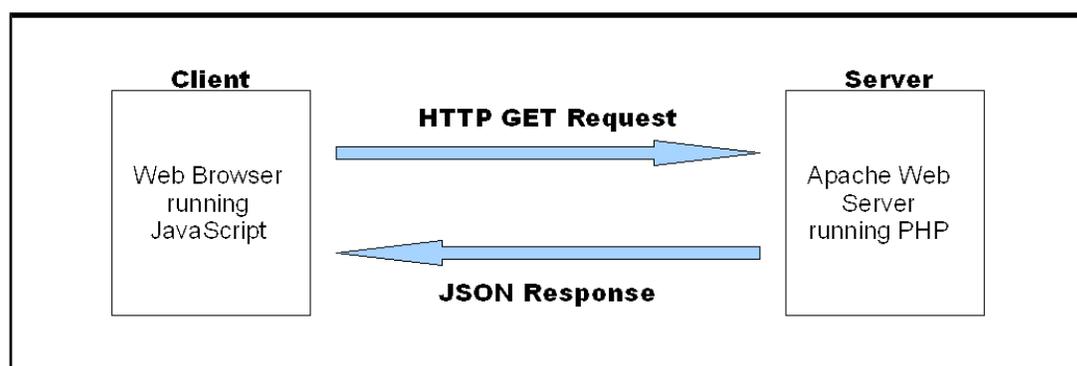


**Figure 11:** Tool button toggle behavior. In the figure, the MoveTool has been activated by clicking on it. This disables some of the other tool buttons from being used. Enabling a different tool button requires first disabling the MoveTool by clicking on it again. This awkward work flow introduces an unnecessary mouse-click.

## GeneJax Implementation

Having considered GeneJax from the user perspective, this section will review GeneJax's internal architecture and lessons learned from its implementation. Just as the preceding section is not a user's guide to the program, this is not an implementation overview. Rather the goal is to assess the history and challenges that arose during GeneJax development since that information will be the most useful to engineers of future genome refactoring tools. The section first explores key client and server design issues and then compares our design decisions with another web based genome visualization tool.

Before discussing design issues further, it is helpful for the reader to first understand the general architecture of GeneJax. The overall server-client structure of GeneJax is shown in Figure 12. Client software runs in the Mozilla Firefox web browser and is implemented in JavaScript. The server runs under the Apache web server and is controlled by a loose collection of Personal Hypertext Preprocessor (PHP) scripts. Most of the application intelligence lies in the client side JavaScript, which was the major focus of our development effort. Given our time constraints, we elected to control project complexity by limiting compatibility to the Firefox web browser; it is known that Internet Explorer and Safari do not work. In contrast, the server side code is simplistic and should run on any web server that supports PHP. All client server communications are transmitted over the HyperText Transport Protocol (HTTP) but the communication format depends on the direction of communication. In particular, client to server queries are HTTP GET requests parsed in an ad-hoc manner, whereas server to client responses are formatted in JavaScript Object Notation (JSON). This communication format asymmetry was initially done for expediency (PHP natively parses unstructured GET requests and JavaScript natively parses JSON) but never corrected to conserve time for other features and because a planned refactoring of the code was canceled as the project fell behind schedule.

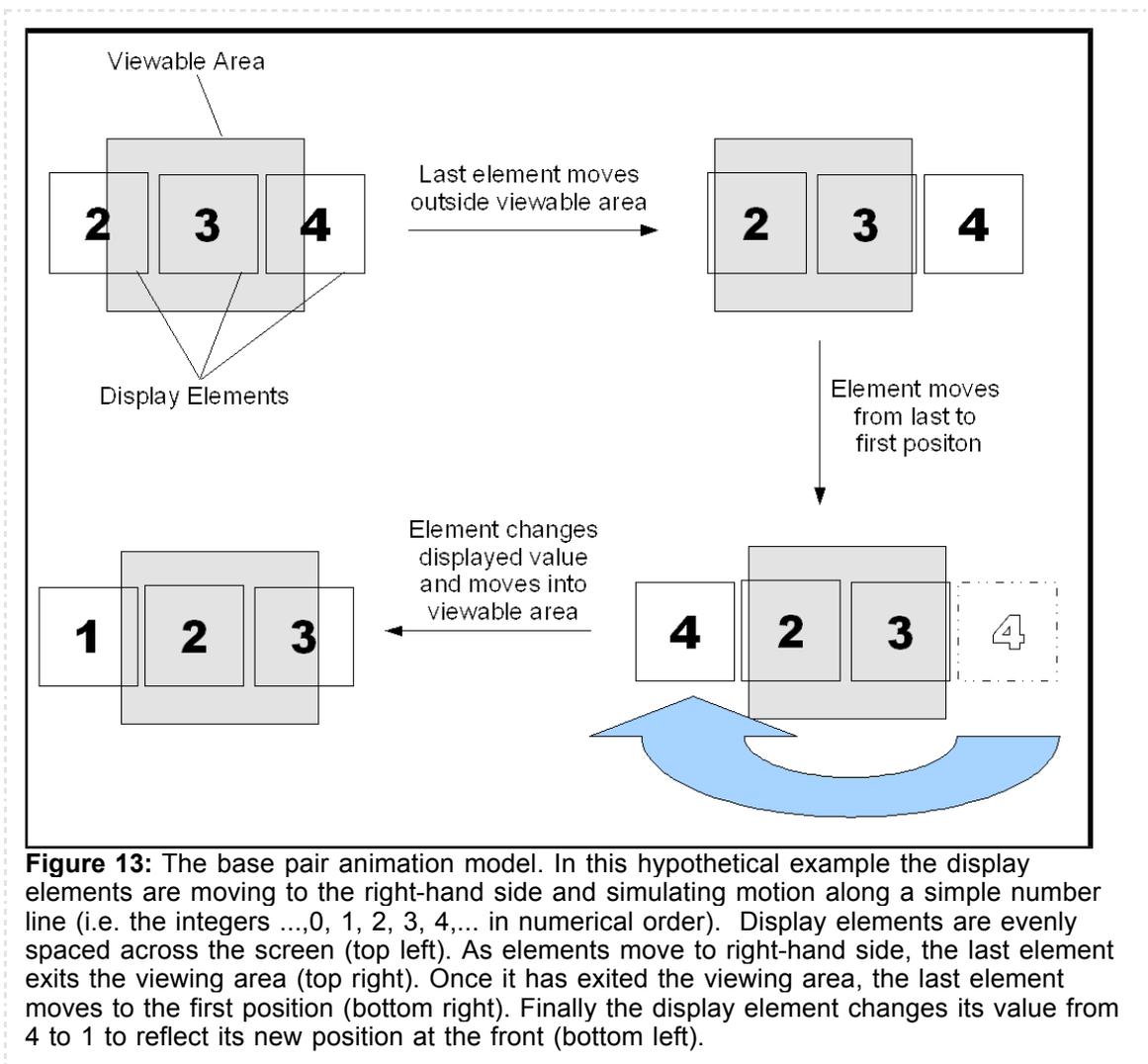


**Figure 12:** GeneJax's Client-Server Architecture. The client runs as JavaScript in a web browser and sends request to the server via HTTP GET requests. The server runs as PHP code on a web server and issues responses in JSON format.

The click-drag functionality central to GeneJax navigation is achieved through JavaScript animation running within the web browser and implementing this animation made up a more significant amount of our development effort than planned, partly due to our development inexperience with JavaScript. In an HTML document, the Document Object Model (DOM) provides JavaScript programs direct control to any HTML element's positional properties, including its absolute horizontal and vertical coordinates on the screen. JavaScript click-drag animation works by changing an element's screen position in response to browser events, such as mouse movements or button clicks. The more elements the JavaScript must explicitly control the slower the animation responsiveness, so the client must limit the amount of animated data. For example, although in theory you could put 40,000+ base pairs of the T7 genome into the

browser and individually animate their movements, in practice the animation would be too computationally and memory intensive to be usable. Instead the client only retrieves and animates data for the area of the genome under inspection. However, due to the tool's development history there are actually two click-drag animation algorithms used in GeneJax.

We started our development by horizontally animating movement along the genome's base pair sequence with click-drag navigation. The animation model was strongly based on the techniques used by two-dimensional image viewers, such as Google Maps and the Giant-Ass Image Viewer (GSV), but now collapsed into a single dimension. Figure 13 shows this animation model as the display elements move to the right. There are a fixed number of display elements evenly spaced across the viewable area. As a display element moves outside of the viewable area, two things happen. First, its position wraps around to the other end of display elements, just outside the viewable area. Second, the display element's content changes to match it's new position. This creates the illusion of scrolling through an infinite row of display elements by using only a fixed number of them.

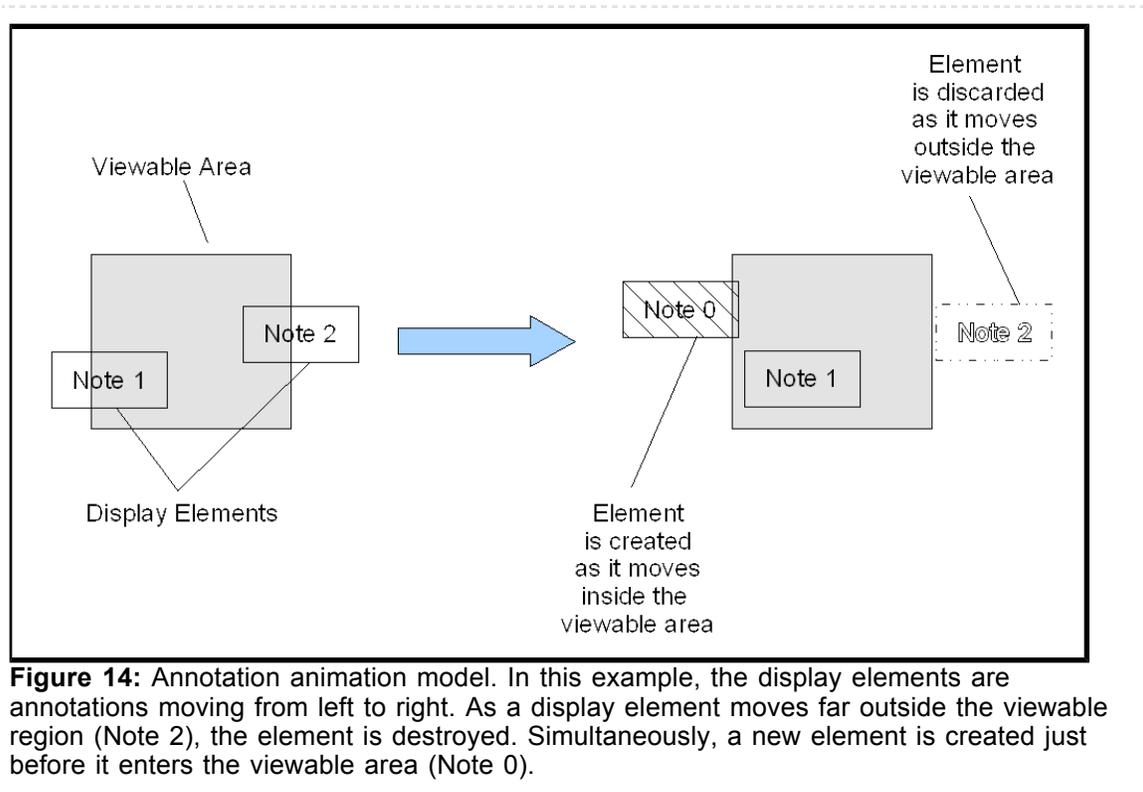


This algorithm was acceptable for animating base pair movement of 10 to 20 base pairs but as we increased the number of elements (i.e. base pairs) animation became sluggish and our CPU

meter showed the machine visibly taxed. Since we had yet to add annotations and parts to the animation, we were quite concerned the browser platform may not be efficient enough to handle the tasks we had planned. Examination of the code for performance optimizations described in (Zakas, 2005) yielded significant gains. Recall that the Document Object Model (DOM) provides JavaScript programs direct control of any HTML element, and that this access forms the basis for JavaScript animation in the browser. Our most useful optimization was to minimize the frequency of these DOM manipulations. During animation our JavaScript code queried the DOM for each HTML element by name before modifying its positional properties. By querying the DOM at the start of the click-drag process and caching the results we were able to remove all DOM operations during the run time of the algorithm. This removed most of the performance bottleneck and made the click-drag animation usable for nearly all practical zoom settings.

It should also be noted that the display elements in Google Maps and GSV are HTML IMG elements. In these tools, different zoom levels change the image data shown by the display elements but the number and size of the display elements remains the same at all zoom levels. In contrast, GeneJax uses text elements from the browser's native fixed-width font as display elements. This allowed us to leverage the font size property in our zoom implementation, allowing more or less base pairs in the screen as the text size is decreased or increased. This is why GeneJax zoom levels are described in terms of font sizes. Although the idea got us up and running quickly, it had a number of unfortunate effects. First, the font size value was not always an accurate indication of a character's true size. Second, it limited the range of usable zoom levels to those at which the browser could display the text font legibly. Third, it limited our display capabilities to pure text so implementing a more graphical (e.g. iconified) view of the genome features would be difficult without making modifications to the algorithm.

A different click-drag animation algorithm was developed when it was realized the initial base pair animation algorithm could not handle annotations and parts. The base pair animation model assumes a fixed number of display elements evenly spaced across across the viewable area. Unfortunately, the number of annotations and parts varies across the genome and is not evenly spaced. In the alternate animation algorithm, the animates each annotation as an independent display element. The client then dynamically creates a new display element for each new annotation as it moves into the field of view, and discards old display elements that have moved far out of view. This process is illustrated in Figure 14 when the display elements are moving to the right. Creating and discarding elements are necessarily DOM operations. Given that the DOM operations formed a performance bottleneck in the first animation algorithm, there was some concern they might do so here too. In practice however this was not the case because creating and discarding elements was a relatively infrequent event. However, in a genome with many more annotations this may be a problem.



Since most of GeneJax's "intelligence" is in the client-side JavaScript, the server software is fairly "dumb". By "dumb" we mean that the server answers stateless queries (e.g. give me the base pairs between positions 100 and 150) without knowledge as to why the client requested something or what it will request next. As a result the server implementation received little attention and was designed for simplicity and minimal development effort. The server software is implemented as loose collection PHP scripts. Each script is designed to answer a different type of query from the client and operates relatively independently of the others. All scripts however refer to two files for their genome data. The first file contains the raw genome sequence in plain text. Because each base pair is a single byte, retrieving the gene sequence for a location is easily translated to an offset in the data file, i.e. the  $n^{\text{th}}$  base pair is located at the  $n^{\text{th}}$  byte in the file. In essence, this turns the gene sequence look up problem into a disk seek problem. Although a compact and fast representation, this would make future editing operations extremely cumbersome (e.g. inserting a base pair into the sequence). Annotations are stored in a separate file as PHP code for an array of structures containing the annotation information. When a script wishes to retrieve annotation information, it evaluates this file into its execution context. This is not a particularly scalable implementation but for T7 (355 annotations) it works fast enough. Again, these scripts were designed for simplicity so we could focus our development efforts on the client software, and in this respect they do their job effectively.

It is useful to compare GeneJax with GBrowse from U.C. Berkeley (Holmes, et. al., 2006). Although designed purely for visualization GBrowse is the only other JavaScript genome web browser of which we are aware. Both projects were proposed independently (and unaware) of each other and took different approaches to implementing genome visualization. GBrowse is more direct translation of Google Maps to genome visualization. Crucially, GBrowse client and server do not do any rendering on the fly. The display elements in GBrowse are HTML IMG elements that are pre-rendered ahead of time and the GBrowse server merely delivers these pre-drawn images which are then positioned by the client. In contrast, the GeneJax client

requests genome sequence data from the server and then renders it into the browser. This confers two key advantages to GBrowse. First, both the client and server have less work to do since all the drawing has been done beforehand, making their implementation less taxing on both the server and client than GeneJax. Second, because it works natively with pre-rendered graphics, the browser can display any kind of graphical "annotation" for a genome feature (e.g. photos, diagrams, etc.). However, despite these advantages it's less clear how part dissection and other editing features could be overlaid onto GBrowse. For example, GeneJax can edit and delete gene base pairs, annotations, and parts because it has rendered each of these items as distinct DOM elements. Every part of the genome that the user sees is independently manipulable by the GeneJax client. For the GBrowse client the base pairs and annotations are inseparably commingled together into static image. Regardless, GBrowse is an impressive, well designed, and arguably better implemented tool for genome visualization. Furthermore, their documentation makes clear the team has considered possibly adding editing and custom annotation features to GBrowse in the future (Uzilov, 2006).

---

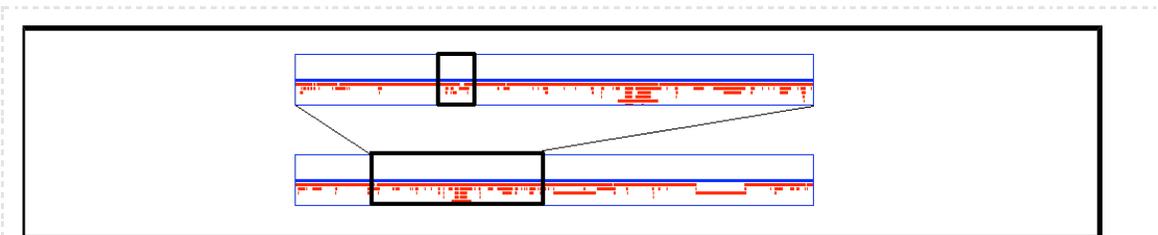
## Insights and Recommendations

Although GeneJax is a usable tool for its intended goal (dissecting T7 sized genomes), it remains a primitive and unfinished experiment in need of further development before it can be useful for a wider audience. In this section we'll discuss proposed features that were dropped from the project, and avenues for improving the visualization, navigation, and usability of GeneJax. Finally we'll consider the ability of GeneJax to scale to larger sized genomes. If the GeneJax project continues these recommendations serve as a starting point for further GeneJax development. Even if GeneJax does not continue, developers of other synthetic biology CAD tools may find that this section offers valuable insights for their projects.

A number of proposed GeneJax features were not completed during the project, partly due to time constraints. Chief among these missing features were cut-and-paste editing of parts and drag-and-drop assembly of a genome from refined parts. These features were dropped when we realized they subsumed the synthesis and part editing functions the MIT Registry of Biological Parts was already developing. Consequently other features that built upon these functions such as a workbench for assembling "super-parts" and editing history were also abandoned. Another proposed feature, integration of GeneJax into the Parts Registry was not finished due to time constraints and logistical difficulties. Given their complementary roles, the most logical integration between the two systems would be to have GeneJax automatically upload newly defined parts directly to the registry. This integration would be highly beneficial, since users currently must import their dissected parts into the Registry by hand.

As described in preceding section, GeneJax's visualization features need serious fixing. The most glaring problem is GeneJax's limited useful zoom range. Although rendering our display elements in the browser's native fixed width font allowed us to get up and running quickly, the first step to resolving our zoom problems should be to modify the rendering algorithms to support images instead of text. By moving our click-drag display elements to images it will be easier to add support for a more "functional" visualization of the genome. In a functional presentation of the genome different annotation and part types are represented by distinct graphical icons. This functional view is a more sensible way to visualize a genome on any scale that shows features larger than hundreds of base pairs in length. Finally, GeneJax's zooming functions should work in concert with the alternative viewing modes, automatically switching between base pair sequence, functional, or a hybrid base pair/functional visualization as the zoom level changes. This is similar to zooming behavior in CLC Bio's Workbench software (CLC Bio, n.d.).

In contrast to visualization, GeneJax navigation is fairly good although still open to enhancements. GeneJax already provides multiple movement modes that let the user easily move either locally (click-drag) or globally (search with auto suggest) through the genome. However, as discussed previously, the slider/map needs further refinement, especially since it can serve as both a visualization aid (conveying positional context) and a third navigational mode (via the slider). One proposed solution would be to make the slider's map support zooming in and out. In this case it would make sense for the map to support the full zoom range (from the entire genome down to individual base pairs), in which case we've turned the map itself into just another viewing pane when it was supposed to provide navigational assistance to the original viewing pane. (Consider for example if the map zooms in further than viewing pane. At that point it provides little navigation assistance.) Another proposed solution is a "multi-resolution" map, mocked up in Figure 15. In this setup a collection of map/sliders is used with each map/slider operating at different pixel to base pair scale. Each map/slider navigates the viewing area for the map/slider above it and the top map/slider reflects the current viewing pane. Unfortunately for large genomes this could mean an unusable number of slider maps. A third option would be a "localized map" that provides a graphical overview of the current viewing location zoomed out by a fixed ratio. For example, the map could always be an order of magnitude zoomed out from the current viewing pane. The localized map seems to be the most appealing compromise since it still manages to provide some positional context without being unwieldy, although the sense of global position in the genome is completely lost.



**Figure 15:** A mock-up of the proposed multi-resolution map and slider. In this example, the bottom map is a map of the whole genome whereas the top map covers a smaller section of the genome. The top map's slider indicates the viewing pane's coverage area while the bottom map's slider places the top map into the context of the whole genome. This design breaks the normally large (i.e. unwieldy) base pair to pixel ratio in a single map into many smaller (i.e. more manageable) base pair to pixel ratios across multiple maps.

Adding forward and back navigational buttons and expanding click-drag navigation to the second dimension should also be considered among the more minor but still potentially useful navigational enhancements. Forward and back buttons similar to those in a web browser could bring sanity to a user that has to hop around between different parts of the genome. Continuing the analogy further, user defined "bookmarks" that take a user to an often used location may also be worthwhile. Also, sometimes the number annotations is so large they do not all fit in the vertical space available in the viewing pane. Users need two dimensional click-drag to view annotations that run off screen. Finally, the biggest implementation challenge to GeneJax navigation may be to ensure existing navigational functionality still works as the visualization features are enhanced.

GeneJax also made some minor, but annoying usability and deployment mistakes that should be corrected. Implementing the tool buttons as toggles proved to be awkward and redundant. Once a user clicks on another tool, they are implicitly disabling the current one. The implementation should be cleaned up so that tools that do not need to be mutually exclusive can be used simultaneously (e.g. click-drag movement and searching) and those tools that truly need to be mutually exclusive (e.g. click-drag movement and part creation), should be selectable without

having to first disable the current tool. GeneJax was also developed and tested solely with the Mozilla Firefox browser. Since the GeneJax project was nominally 96 hours (6 hours per week for a single semester) this was an acceptable compromise to save precious development time. Moving forward cross-browser support is crucial for the success of the GeneJax project. Any browser application that does not work with the dominant web browser will be marginalized.

GeneJax will also require further changes to support larger sized genomes. On the server, our method for base pair sequence retrieval will scale to larger genomes without difficulty, but the annotation retrieval method will not. In addition, these methods store their genome data in non-standard files that are unwieldy to modify. Finally, the genome data stored is stored as plain files and primitive file system protections are the only mechanism for managing multiple users reading and writing the data. Migrating the genome data to a full fledged database could solve these problems.

In contrast to the server side, most of the GeneJax client code should scale to larger genomes although there are some areas for concern. Because the client stores only a local subset of the genome data, the size of the genome isn't an issue per se. Instead, the crucial weak point is the number of annotations a genome may have in any given region. In the current implementation each base pair and annotation is an individual animation element, and each independent animation element imposes a computational overhead. Too many annotations leads to too many animation elements which can swamp the animation algorithm. One way to mitigate this problem might be to dynamically combine elements together just prior to animation. For example, each base pair and its corresponding base pair on the complement strand will always move in lock step, so they could be combined into a single animation element. In this manner the number of animation elements might be reduced to a more manageable level. In the same vein, performance gains could be achieved by migrating the base pairs to the annotation/part animation algorithm (e.g. where each group of 5 base pairs would be considered a single "annotation"). This would also reduce the maintenance headaches of having two different animation algorithms instead of one. Last but not least, the implementation would be cleaner if both client to server and server to client communication standardized on the same communication format, i.e. JSON.

---

## Conclusions

This paper has described the GeneJax prototype program. We started by introducing genome refactoring and positioned GeneJax as a tool for the dissection stage of this process. Next, we reviewed the program's end-user the interface and assessed the interface's features and flaws. We also discussed the software implementation and the merits (or lack thereof) of GeneJax's internal design. Finally we made recommendations for future GeneJax development. These recommendations will be necessary to move GeneJax from a prototype to a practical and deployable program.

Looking forward, GeneJax's future is questionable for two reasons. First, GeneJax is still a prototype. Significant design and engineering is still required to turn it into practical tool for a wider audience. Second, the GeneJax project was designed to address an unmet need in the refactoring process. It is not unimaginable that this gap could be closed as existing, more entrenched, and better funded projects expand to fill GeneJax's purpose. For example, GBrowse and the MIT Biological Parts Registry border GeneJax on either side of the refactoring process (GBrowse for visualization, the Registry for part editing and synthesis). Both projects are under active development and could subsume the dissection stage as genome refactoring becomes more widespread. Since it deals in genetic parts, dissection is arguably under the mandate of the Parts Registry. Meanwhile GBrowse may prove to be the more scalable visualization

architecture, and if so, it would make more sense to layer a dissection tool on top of it. Like any program, GeneJax development should only continue if it fills a need and there is good reason to believe it will fulfill that need well. Regardless of GeneJax's fate, we hope that designers of future genome refactoring tools will find the information here useful.

## Bibliography

Chan, Leon Y. Kosuri, Sriram. Endy, Drew. (2005) Refactoring bacteriophage T7. *Mol Syst Biol* 13 September 2005; doi:10.1038/msb4100025

CLC Bio. (n.d.) Tutorial: View Sequence. Retrieved May 16, 2006 from <http://www.clcbio.com/examples/pdf/build-tutorial-viewsequence.pdf>

Fowler, M. Beck, K. Brant, J. Opdyke, W. Roberts, D. (1999) Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley Professional

Garret, Jesse James (2005) Ajax: A New Approach to Web Applications. *Adaptive Path, LLC*. 18 February 2005.  
<http://www.adaptivepath.com/publications/essays/archives/000385.php>

Google Finance (n.d.) Retrieved May 15, 2006 from <http://finance.google.com>

Google Maps (n.d.) Retrieved May 15, 2006 from <http://maps.google.com>

Holmes, Ian. Mungall, Chris. Uzilov, Andrew. (2006) Prototype G Browse. Retrieved May 15, 2006 from <http://biowiki.org/view/GBrowse/PrototypeGBrowse>

Lewis, Christopher T. Karcz, Steve. Sharpe, Andrew. Parkin, Isobel A.P. (n.d.) BioViz: Genome Viewer. Retrieved May 15, 2006 from <http://spark.sourceforge.net/resources/samples/bioviz/>

Migurski, Michal. (2005) Giant-Ass Image Viewer. Retrieved May 15, 2006 from <http://mike.teczno.com/giant/pan/>

MIT Registry of Standard Biological Parts (n.d.) Retrieved May 15, 2006 from <http://parts.mit.edu>

National Center for Bioinformatics (n.d.) Retrieved May 15, 2006 from <http://www.ncbi.nih.gov/>

"T7.2" (2005). <http://openwetware.org/index.php?title=T7.2&oldid=12368>

UCSC Genome Browser (n.d.) Retrieved May 15, 2006 from <http://genome.ucsc.edu/>

Uzilov, Andrew. (2006) Prototype Genome Browser. Retrieved May 15, 2006 from [http://biowiki.org/twiki/pub/GBrowse/PrototypeGBrowse/03-17-2006\\_\\_prototype\\_genome\\_br](http://biowiki.org/twiki/pub/GBrowse/PrototypeGBrowse/03-17-2006__prototype_genome_br)

Wikipedia contributors (2006). Refactoring. *Wikipedia, The Free Encyclopedia*. Retrieved 20:25, March 8, 2006 from <http://en.wikipedia.org/w/index.php?title=Refactoring&oldid=41355008>

Zakas, Nicholas C. (2005) Professional JavaScript for Web Developers. Indianapolis, IN, USA: Wiley.

Zakas, Nicholas C. McPeak, Jeremy. Fawcett, Joe. (2006) Professional Ajax. Indianapolis, IN, USA: Wiley.