CYTOSCAPE BOARD REPORT

OCTOBER 31, 2007

EXECUTIVE SUMMARY

With the wealth and depth offered by competing products, Cytoscape will need to adapt to ensure that it can remain a useful tool for the scientific community. The direction and the extent of this change is a decision that the board needs to make. This document summarizes a special meeting of developers that was held on October 11-12, 2007 at the ISB, and presents the findings of what would be required in three general areas: developing an architecture to improve Cytoscape; changing the plugin mechanism to allow for maintainability; and improving integration. Where it was appropriate, the choices (involving scope, prioritization and methods) that can be made about these areas are highlighted, and a summary is presented at the end of the document. By the board discussing and agreeing on these issues, it will be possible for the Cytoscape development community to move forward in a unified manner towards a central goal.

EXECUTIVE SUMMARY

With the wealth and depth offered by competing products, Cytoscape will need to adapt to ensure that it can remain a useful tool for the scientific community. The direction and the extent of this change is a decision that the board needs to make. This document summarizes a special meeting of developers that was held on October 11-12, 2007 at the ISB, and presents the findings of what would be required in three general areas: developing an architecture to improve Cytoscape; changing the plugin mechanism to allow for maintainability; and improving integration. Where it was appropriate, the choices (involving scope, prioritization and methods) that can be made about these areas are highlighted, and a summary is presented at the end of the document. By the board discussing and agreeing on these issues, it will be possible for the Cytoscape development community to move forward in a unified manner towards a central goal.

INTRODUCTION

This report is for the Cytoscape board members and is designed to convey the alternative strategies that are available for the proposed software rearchitecting. The advantages and disadvantages of each of these strategies are presented, along with dependencies between them. Where appropriate the justified recommendations of the meeting are also given.

To go forward with the rearchitecting, there are a number of choices in a number of areas, and this document is designed to ensure that the board is able to make well informed decisions about which strategy they wish to adopt.

The first section of the report presents an overview of the architecture meeting that was held. The second section of the report outlines the different strategic areas that were identified as being important in a rearchitecting effort, along with the alternative choices and suggestions from the developers. The third section presents a summary of these findings. In the appendix, the relevant RFCs, with project plans, and other relevant documentation is provided.

MEETING REPORT

On October 11-12, 2007, a Cytoscape developer team (Allan Kuchinsky, Sarah Killcoyne, Mike Smoot, Alex Pico, Scooter Morris, and John Boyle) from several member institutions, met to discuss a rearchitecting of Cytoscape. The first part of the meeting was given over to requirements gathering in two important areas:

- **Due Diligence on selected competitive products.** Several network visualization products were assessed by the attending developers. These products were chosen so as to encompass both a wide range of usage and features. There were many features in the other products that would make Cytoscape both more useful and more usable, including: simple features (e.g. generic desktop operations such as copy paste); high levels of integration (e.g. with tools like Matlab and R), advanced server based functionality (e.g. having a knowledge base); and alternative delivery mechanisms (e.g. both automation and web delivery). Some of these features could only be effectively implemented as part of a rearchitecting of the current software.
- **User feedback discussions**. We heard from three different types of users that are representative of ISB users: a computational biologist, a plugin developer and a general biologist. Several issues were identified as preventing potential and previous users from using Cytoscape as a part of their research. These included the lack of backwards compatibility of plugins to basic instabilities and problems due to lack of reliability, robustness and documentation.

Based on the above discussions and an overview of the RFCs, we then drew up a list of required features, and then used these to shape the discussion on what would be required to move Cytoscape towards a componentized software framework that supports the flexible, and often ad-hoc, extensions to the functionality that are required in the life sciences.

MEETING OUTCOMES AND RECOMMENDATIONS

After discussing the requirements, the potential solutions, along with strategies for achieving them, were discussed. The solutions obviously have a high dependency, and can be summarized as the following features requests:

1) *Rearchitecting*. General purpose rearchitecting that will move Cytoscape from the current monolithic tightly coupled codebase to a modular and modern software architecture. Two specific features are currently lacking: Resource Management – uniform mechanism for finding and limiting resources (threads, memory and connections) and Event and Exception handling – models to ensure uniform propagation and triggering of events and errors.
2) *Plugin Architecture*. Ensure that new plugins are backwards compatible and can be tested. This requires the development of both an API for Cytoscape so that plugins can call methods in a uniform, consistent and maintainable manner; and a plugin API so that Cytoscape can invoke desired functionality on the plugin. The actual mechanism(s) for these systems can involve any of: interface development, service development, or aspects development.
3) *Integration Tools*. Sets of tools that will attract new users, help users to develop better plugins and provide all users with a system to organize their data. A few of the ideas included: creating Matlab and R toolboxes (for computational biologists) to create network files in formats Cytoscape expects; adding a project system to organize and display the various types of data that belong to a network; and plugin development tools (from how-to's to templates) to aid developers in using the plugin architecture and various tools (e.g. Layouts, VizMapper).

REARCHITECTING CYTOSCAPE

Ultimately the goal of Cytoscape is to provide a tool that helps scientists to do their work better through the use of visualization and analysis tools and supporting community development of plugins scientists can use. In order to meet that goal, Cytoscape needs to be updated through a rearchitecting of the current code. This would help Cytoscape users to: create more and better plugins; support backwards compatibility of plugins; support easier data integration; and have better support for collaboration.

There are a few options for how we can rearchitect Cytoscape. Any choices we make about new features to implement will depend on which method we choose for this rearchitecting: refactor, taking the current code and reorganizing it as best as possible; or rewrite (there are two ways we can rewrite as well), utilizing the Cytoscape developers' experience in solving the important problems of Cytoscape to start with a clean slate.

STRATEGY 1: REFACTOR - 18 MONTHS, 1 FTE

Advantages: (i) The code will be changed incrementally meaning it should be useable throughout the process making it somewhat less risky to do. **(ii)** The current Cytoscape method of working remains the same (small distributed development teams, with little project coordination required).
Disadvantages: Old solutions may prevent clean and incremental transitions making the end result less likely to meet design goals.

This option takes all of the current code in Cytoscape and reorganizes it, slowly. Dependencies are first encapsulated within an additional set of classes, and then refactored out into a more logical and uniform class structure. This will require extra interface work to separate out the current dependencies of Cytoscape, but will ultimately allow for the required modular desktop system. There are several new features that should be introduced during this type of refactoring:

- ***Event Handling***. Under a refactor, a queue system may be used to register plugins that are interested in particular events and running them in order. Alternatively, instead of permitting all plugins to listen at the same time the user could choose or "focus" the plugin they want to use and only that plugin will get the event.
- ***Resource Management***. We should also provide a resource management scheme for thread, memory and connection management. This would involve the specification and formalization of the interfaces to these managers, as well as the implementation of the underlying managers themselves. In some cases, such as connection management, appropriate parts of other technologies (e.g. Swing) could be used.
- ***Exception Handling***. Define exceptions for each layer and create a guideline for the various types of exceptions (errors, exceptions that should be caught versus passed along) and add logging. This would be a good use of aspects.
- ***Command Layer.*** A command layer could be introduced to enable the adoption of a scripting mechanism. Such a command layer would translate each command into an object (or string) representation and allow for both its recording, and possible the use of command history stacks (therefore supplying a convenient undo/redo mechanism).

STRATEGY 2: REWRITE- 6 MONTHS, 3.5 FTE

Advantages: (i) It will be easier than a refactor because we will not be tied to old solutions and we already have a full specification. **(ii)** We will not be limited by our current design, and so can adopt a more radical architectural change (e.g. move towards a thin client architecture).
Disadvantages (i) Riskier because there are no completed incremental products to show and use **(ii)** Would require a fundamental change in the way in which Cytoscape is developed (e.g. a small closely connected core team would be required).

If we consider a rewrite then we are no longer restricted to the current architecture, and so can adopt either the same architecture or a distributed/thin client system:

- ***Strategy (2a): Rewrite Fat Client - 6 months, 3.5 FTE***. A full rewrite of the current desktop environment would allow us to take advantage of the experience of the full core team and start with a clean slate to correctly handle overriding issues such as error handling, event handling, and complete unit testing as well as freeing up developers from relying on old interfaces (or spaghetti code). All of the base features detailed in the refactor (above) would be handled the same way in a rewrite. This means that a rewrite would take less time.
- ***Strategy (2b): Rewrite Thin Client and Server - 6 months, 3.5 FTE***. A full rewrite could also entail the development of an n-tier, thin client application. This option will enable the reuse of code from a number of enterprise computing technologies (e.g. Tomcat). This is similar to what the Broad Institute at MIT did for their Gene Pattern software, which can be run as a local desktop GUI or through a server for access to more powerful functionality. The advantage of such an architecture is that there is a good return on investment, as required functionality can be utilized from 3rd party application servers, such as: resource management, state, messaging, event handling and data management. Additionally, we can introduce "enterprise" functionality rapidly through the (re)use of clustering, querying, messaging (event handling), logging and integration technologies. The major drawback of this approach is that it is a fundamental change to Cytoscape code and, more importantly, the development process. Such a change makes risk management harder than with rewriting the Cytoscape client as-is.

PLUGIN ARCHITECTURE

Cytoscape's greatest strength has been the functionality offered through plugins created by a variety of other groups. It has allowed Cytoscape to extend functionality that would have been difficult for a small core team to support. However, as the software has evolved and more plugins have been written, the lack of a defined plugin operation specification has become a problem. Due to the tight coupling of the plugins to core code it is currently possible for a plugin to conflict with core processes and other plugins. This issue can be corrected with the rearchitecting of the core.

Once the specification and preliminary work on rearchitecting has been completed, a plugin architecture can be introduced. This can be best achieved by introducing specified interfaces. These interfaces would basically be a contract for the different layers of Cytoscape. A more robust, and portable, solution could then be introduced through the use of dependency injection, possibly through the introduction of aspects. A proper event handling system and resource management in the core would also improve the new plugin architecture.

If a distributed system is put in place, the plugin architecture would be slightly different. Plugins would have to reside on the client or the server (POJO detachment could be used, but would be ill advised). Client code would remain the same, but server code would follow the application server specification. In this way, plugins become essentially widgets or services through using well-known java technologies such as servlets, Spring, or even portlets.

INTEGRATION SYSTEMS

To help expand the user base, a number of integration methods and tools were also highlighted. Most of these depend on completing a refactor/rewrite and others may depend on specific features within the options listed above (such as a new plugin architecture).
These integration systems are:

- **Project System.** We can implement a project system through the addition of a state system and the use of swing models. The model behind a project system would be easily serializable for simple development and the idea of a "Project System" is very natural to the web environment so a lot of the backend work may already be available to us through 3rd party code potentially making this very fast to implement.
- **Plugin Development Kit (PDK).** After a plugin architecture has been added to Cytoscape, a development kit to assist plugin developers to write and test Cytoscape plugins would be extremely useful in encouraging current and new plugin writers to write or rewrite plugins.
- **External Tool Development.** A number of tools should be developed to make integration with Cytoscape easier. Principally toolboxes written for Matlab (and R) users to save networks in XGMML or to push data directly to an open Cytoscape session would greatly encourage and assist Cytoscape use by computational biologists. This feature could be added any time with the help of people knowledgeable in Matlab/R and XGMML.
- **Domain Specific Models.** Specialized plugins that overlay the basic network model of Cytoscape with a domain specific model such as genes, proteins or molecules to help encourage various disciplines to use Cytoscape by offering them vocabularies that are familiar to them.
- **Project Editors.** Most tools that incorporate a project system also have some editing tools for their projects. Tools that allow data to be cut copied or moved to other projects as well as tools to assist in the creation of a new project (e.g. wizards). This particular feature could be written in tandem to the project system.
- **Web Front End.** Many of the most successful tools of all types offer some ability to use them online. Cytoscape could reach many new users who may have little interest in installing a new application, but can easily use a browser. This would greatly facilitate sharing of networks and the general team approaches common to systems biology.
    - ***Rewrite/Refactor Desktop Client.*** This would take about 16 weeks to complete after the core code has been refactored and it would ultimately entail doing most of the work necessary to create a thin client.

- ***Rewrite Server Client.*** This would take about 8 weeks to complete after the thin client has been written. Technologies such as AJAX, SVG, Google Maps or portlets could be used.

OTHER ISSUES

In our discussions we heard a lot of user feedback that amounted to needing more help for the basic user and plugin developer. The best solution to such needs is what we called a Cytoscape "evangelist". This would be someone who could attend to: marketing Cytoscape to help attract new users; keeping up user documentation and how-to manuals to make the learning curve a little gentler; answering the help desk or shunting those questions to the appropriate people within the team; organizing user meetings like the one in Seattle two years ago; helping to make sure errors that are really frustrating users get attention. The GenMapp project was mentioned as an example of this. GenMapp has a dedicated person to handle these sorts of issues and she has been credited with a lot of the success of that project. This would help Cytoscape to grow by attracting and helping new users to discover and use the software.

SUMMARY

RELATED DOCUMENTS

Listed under Appendix 1:

- 1-A: October Seattle Meeting Agenda
- 1-B: RFC Refocusing Document

Listed under Appendix 2:

- Included RFC's:
  1. 2-A: Layering of Cytoscape Code
  2. 2-B: Cytoscape Rewrite (includes Fat Client and Thin Client options)
  3. 2-C: Plugin Architecture Refactor
  4. 2-D: Event Handling
  5. 2-E: Resource (Task) Management
  6. 2-F: Cytoscape Projects
  7. 2-G: Web Front End
  8. 2-H: Domain Specific Models: Cytoscape "Themes"

SUMMARY TABLES

The following tables layout the possible routes for rearchitecting Cytoscape and additional tools that were considered useful. The sections have been broken down by the refactor or rewrite options with the features that are part of each and estimated times to develop. Based on the developer discussions priorities have been added to each, 1 (highest) - 5 (lowest) and a column has been left open for reprioritizing based on board discussions.

REFACTOR

See RFC: Cytoscape Layers Refactor

Project Name	Prior Dependency	Est. Time (months)	Priority	Note	Board Priority
1. Complete Refactor	n/a	18mo 1 FTE	1		
Event Handling	n/a	3mo 1 FTE	2		
Exception Handling	n/a	2mo 1 FTE	2		
Resource Management	n/a	3mo 1 FTE	3		
Command Layer	Complete Refactor	4mo 1 FTE	5		
Plugin Rearchitecture	Complete Refactor	6mo 1 FTE	2		

REWRITE

See RFC: Cytoscape Rewrite

Project Name	Prior Dependency	Est. Time	Priority	Note	Board Priority
2a. Fat Client Rewrite	n/a	6mo 1 FTE	1		
Event Handling		2mo 1 FTE	2		
Exception Handling		2mo 1 FTE	2		
Resource Management		2mo 1 FTE	3		
Command Layer	Fat Client Rewrite	2mo 1 FTE	5		
Plugin Rearchitecture	Fat Client Rewrite	6mo 1 FTE	2		

2b. Thin Client Rewrite	n/a	7mo 3.5 FTE	1		
Event Handling		n/a	2	*	
Exception Handling		n/a	2	*	
Resource Management		n/a	3	*	
Command Layer	Thin Client Rewrite	3mo 1 FTE	5		
Plugin Rearchitecture	Thin Client Rewrite	4mo 1 FTE	2	**	

* Tomcat handles this for us.
** This estimate is for visual plugins that would use the desktop application client.

INTEGRATION TOOLS

See RFC's: Cytoscape Projects, Web Front End for Cytoscape, Domain Specific Models

Project Name	Prior Dependency	Est. Time	Priority	Note	Board Priority
Project System	Rewrite/Refactor	*	3		
Project Editors	Project System	*	4		
Web Browser Front End	Rewrite/Refactor	*	3		
Domain Model Plugins	Plugin Rearchitecture	*	5		
Plugin Development Kit	Plugin Rearchitecture	*	3		
Matlab/R Toolboxes	n/a	*	4		

* Timeline depends on which rearchitecting option (refactor/rewrite) is chosen.

APPENDIX 1

1-A: OCTOBER SEATTLE MEETING AGENDA

CYTOSCAPE MEETING OCTOBER 11-12

Goal: Put together a "shopping list" of ideas to present to the board regarding re-architecting Cytoscape. To accomplish this we want to keep this discussion high level, and so we shouldn't discuss code or specific releases directly as those details should continue to be handled by the core development group as a whole.

THURSDAY OCT. 11

9:20AM

- Lee Hood - Welcome
- Ilya Shmulevich

9:30AM

PRODUCT ASSESSMENTS

- Each package will have a brief write-up:
  - Brief description of the package
  - Target audience/primary use case
  - Last update
  - Plugin architecture
  - List of features that we like (even if Cytoscape implements it as well)
  - List of features not available in Cytoscape that might be useful

Packages

- Allan - PathwayArchitect
- Mike - Biological Networks
- John - VisAnt
- Scooter - Osprey
- Alex - Ingenuity
- Sarah - Pajek, CellDesigner

10:30AM

DISCUSSION: DUE DILIGENCE

- What do we like about these applications? Specifically look at the write-ups
- What have they done well, what might be useful to our users?

12:00PM

Lunch

1:00PM

USER EXPERIENCES

- Greg Carter – “issues with maintaining our plugins”
- Ilya Shmulevich – “issues with use of Cytoscape by computational biologists”
- Jen Smith - “issues with teaching Cytoscape”

1:30PM

DISCUSSION: CAPTURE PROBLEMS WITH CYTOSCAPE

- Based on user experience and application discussions from the morning session

3:00PM

DISCUSSION: RELATED DEVELOPMENT EXPERIENCES

- Sarah Killcoyne – integrating distributed ad-hoc data services
- John Boyle – past experiences in managing the rearchitecting of mature products
- Hector Rovira – software process used to refactor a plugin architecture into an existing commercial product

4:00PM

DISCUSSION (FOR LEAD IN TO FRIDAY)

- Review identified problems
- Discuss potential solutions

5:30PM

Dinner & drinks

FRIDAY OCT. 12

9:00AM

DISCUSSION: PRACTICALITIES OF THE CYTOSCAPE DEVELOPMENT CULTURE

- Requirement for short iterations
- FTE numbers and distributed development
- Plugins

9:30AM

OVERVIEW RFC’S

- Break down the available RFCs into functional sets

10:00AM

DISCUSSION: REVIEW THE RELEVANT RFCS AND RELATE TO IDENTIFIED PROBLEMS

- Match problems identified on Thursday with RFCs
- Highlight problems which are not discussed in RFCs

12:00PM

Lunch

1:00PM

PLANNING BASED ON PROBLEMS AND GROUPED RFC'S

- Present high level ideas for solving the problems: including strategies for implementation, preliminary timelines, problems and dependencies
- Establish groups to work on adapting old (or producing new) RFCs
- Put together "shopping list" of ideas to present to board

4:00PM

Departure

1-B: RFC REFOCUSING

DUE DILIGENCE FOCUS

Importance: 1 (most) - 3 (least)
* 3.0+ issues

MODEL

1) Project-domain models*
    - Must have a view
    - Potential to request an update?

1) Themes*
    - Domain model overlays (genes, proteins, etc.)
    - Annotation standards

3) Add Z and T to model views?*

IO

1) Clipboard data flavors
    - Specifically cut-and-paste for Excel

1) Generalize attribute input mechanism
    - Target microarray files

2) R/Matlab toolbox XGMML
    - Other tools? SBW?

CONTROLS

2) Inventory controls
1) Task monitoring & cancelable threading and view*
1) General exception/error handling*
    - Submit bugs on uncaught exceptions
2) Installation configuration wizard
2) Auto-updates of Cytoscape

VIEW

1) Web Front End*
1) Publication-quality images
    - Background graphics
1) Themes*
2) Multiple views*
1) Macros*
    - Protocol interface
    - User-created workflows
3) Boxing nodes/Bubble router
3) Animated layouts
3) Pre-filter large network for view via Matlab toolbox/external utilities

PLUGIN

1) Consistent event handling (in cytoscape) – expose “integration points”*
1) Backwards interoperability*
1) Plugin Interoperability*
1) Resource management*
2) Test frameworks*
2) “Scripted” plugins*
3) Consolidated network analysis plugins
    - Toplogy searching
    - Clustering
    - Network statistics

DATA INTEGRATION

1) Pulling content & presenting
    - GeneInfo
    - Networks/pathways
3) Data “knowledge base”*
    - Must be local?
    - Mediator, plugin
3) Project server *
    - Storing networks, attributes, etc. in a shared central database
3) AOP data integration*

APPENDIX 2 - REFERENCE RFC DOCUMENTS

2-A: LAYERING OF CYTOSCAPE CODE

Sarah Killcoyne, John Boyle, Mike Smoot
August 14, 2007

PROPOSAL

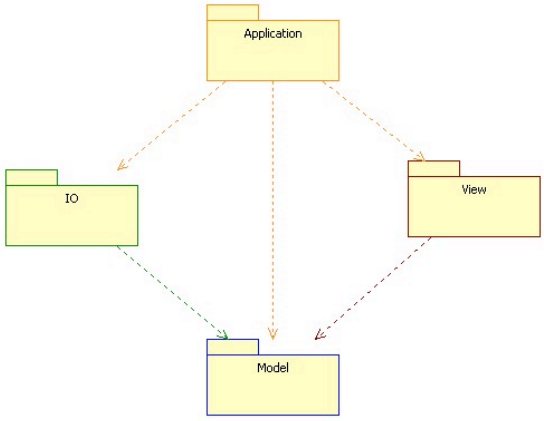
Many Cytoscape clones are available that: utilize both web and server based technologies; use advanced information visualizations of graphs; and support a plugin architecture. While Cytoscape is still preferentially, we can improve the functionality in a number of ways, including: clear scripting/macro capability; web based delivery; allowing for high throughput asynchronous message based analysis; server/distributed functionality; uniform access to data sources; and a componentized code base (e.g. to change the rendering code). While it may be possible to do many of these things with the current code base, all of these would be simpler and cleaner to implement if the code is relayered to separate the functionality. This would also enable cleaner builds and ensure that the code is maintainable in the future. This proposal builds upon the following RFCs: Code Layering[38], Scripting in Cytoscape[40] and Cytoscape Headless Mode Operation[6].

In order to accomplish the goal of relayering we will introduce use cases/features, including those that that require scripting, web front ends and easier maintenance. We will also discuss the package structure and build dependencies. The plan is to iteratively move towards a fully layered architecture through meeting several milestones that will also allow Cytoscape to continue some forward development during this work. The current package structure will first be refactored into the an intermediate structure that mimics that model/view relayered code we desire, then a set of temporary interfaces will be created to handle the dependencies that will exist to allow us to meet the first milestone. We can then take that structure and set up each part as an OSGi service with the registry to meet the second milestone. Finally we can go back and remove the various artificial interfaces and dependencies we introduced initially in order to have a fully layered system. At this point it will be much simpler to set up interfaces for plugins, scripting and command-line control of Cytoscape.

USE CASES

- Allows for headless mode/scripting/macros to be implemented fairly simply
- Easily change/replace giny backend with other rendering code
- State capture
- Clear logging
- Switch around front ends: swing gui/Spotfire/Web site
- Portlets
- High throughput analysis (via headless/server/scripting)
- Better plugin interfaces using each layer only as required
- Distributed/Server based functionality
- Robust code
- Easier maintenance and testing

BUILD DEPENDENCIES



The proposal is to initially introduce three layers into the code: Model layer which contains the business logic; view layer containing all the graphical/presentation code; and the IO layer which controls readers/writers. These will be built as subsystems, with the no dependencies existing on the model layer. View and IO systems will be dependent on Model and Application will depend on all three. Temporarily we will introduce a "Common" subsystem which all other systems will depend upon to allow us to factor out dependencies more simply. As the relayering progresses, interfaces from the Common system will be refactored to remove this dependency.

PACKAGE STRUCTURE OVERVIEW

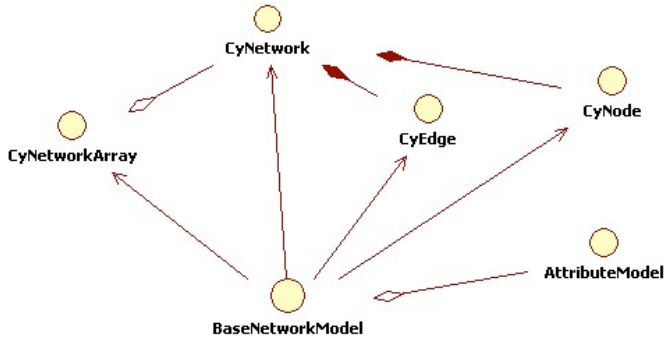
The packages will be broken into four main groups as described above: *model*, *view*, *application* and *io*. Each of these packages will contain classes specific to that part of the system.

- Model will contain two main packages: *network*, for the model objects required to create a network and handle network events and *attribute*, for the model objects required to create attributes and handle attribute events. The model package will have no dependencies outside of itself (except temporarily the *common* package).
- View will contain two main packages: *network*, for viewing the entire network at the macro and micro level (and any other views we may wish to add to a network); *attribute*, for viewing object attributes using tables, notes or other views.
- IO/Comms will again contain the two packages *network* and *attribute* for the finders and producers of the two *model* objects, initially just filesystem based, this can eventually be expended to handle database or internet finders and producers.
- Application will contain several packages that will be specific to the swing gui application including: *dialogs,* to contain all of the various dialogs required for user interaction or messaging; *init,* to set up and start the application; *process,* to handle the tasks various parts of the application need to do; *widgets,* for bits like the visual styles and menu actions; *util* for any utilities package necessary.

MODEL LAYER

The model layer contains the basic data structures necessary for the underlying graph, including networks, nodes, edges and attributes.

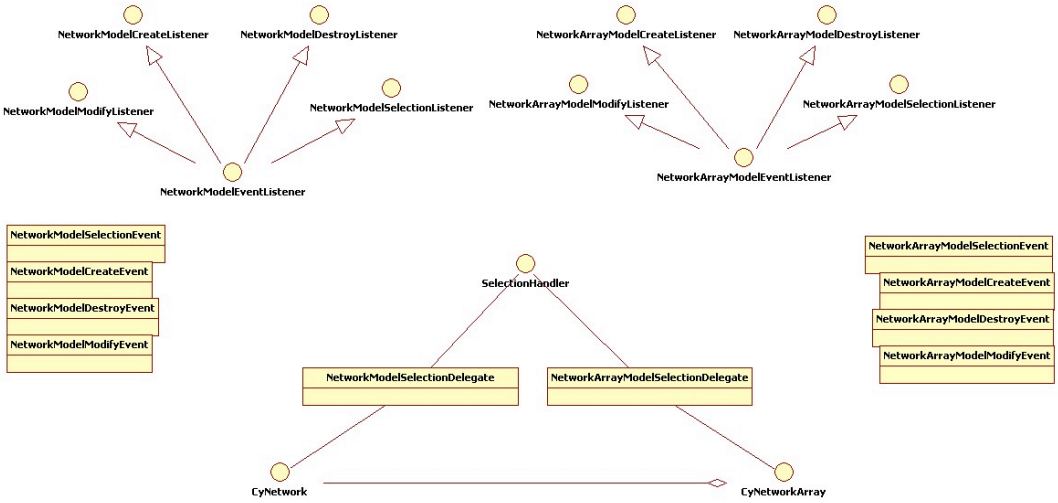
MODEL DEFINITION



The **network** package structure will be a set of interfaces for the basic network object types. All will inherit off of a single interface that will contain basic information each network object should need (see below).

- *BaseNetworkModel*: interface for the basic information an object in the network package should contain including unique identifier, name, attributes and selection state
- *CyNetwork*: interface (extends *BaseNetworkModel*) for additional requirements of a network such as creating, adding and removing a node/edge or groups of nodes/edges
- *CyNetworkArray*: interface (extends *BaseNetworkModel*) for requirements of lists of networks such as adding, removing and getting a *CyNetwork* from the list
- *CyNode* and *CyEdge* are interfaces (extending *BaseNetworkModel*) for the Node and Edge object types

MODEL EVENTS

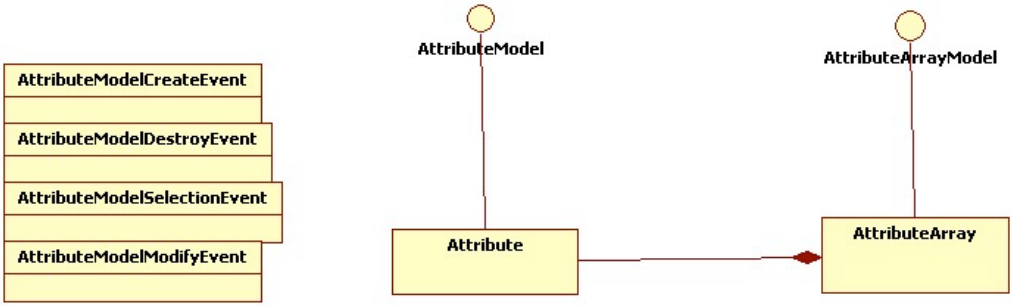


The **network event** package structure is based on the standard java event pattern (<item>producers firing <item>events to a list of <item>consumers). The definition and reason of the event objects being past between objects depends on the type of event, and is detailed in the

description below. Each model contains a list of its consumers, however to enable the ability to have multiple selections propagation mechanisms attached to one model a series of selection delegates can be attached to the model (each containing separate lists of listeners.

- The *CyNetworkArray* can fire the following events to the related listeners:
  - *NetworkArrayModelSelectionEvent*: Contains a set selection object containing the identities of the network models that have been selected, the reason can be one of *SELECTED* or *DESELECTED*.
  - *NetworkArrayModelCreateEvent*: Contains the identity of the new network model that has been created.
  - *NetworkArrayModelDestroyEvent*: Contains the identity of the network model that has been destroyed. This could possibly be a *vetoable* action.
  - *NetworkArrayModelModifyEvent*: The reason can be one of *NAMEMODIFY*, *STRUCTUREMODIFY*, *IDMODIFY*, *ATTRIBUTEMODIFY*. The *NAMEMODIFY* and *IDMODIFY* means that the string representations have changed, *STRUCTUREMODIFY* means that the order of models has changed and the *ATTRIBUTEMODIFY* means that the attributes/descriptions associated with the *NetworkArrayModel* has changed.
- The *CyNetwork* can fire the following events to the related listeners:
  - *NetworkModelSelectionEvent*: Contains a list of the edges and nodes that have been selected. The reason can be one of *SELECTED* or *DESELECTED*.
  - *NetworkModelCreateEvent*: Contains a list of the new nodes or edges that have been created. The reason that is passed with the event is one of *NODECREATE*, *EDGECREATE* or *NODESANDEDGESCREATE*.
  - *NetworkModelDestroyEvent*: Contains a list of the nodes or edges that were destroyed. The reason for the event can be one of *NODEDESTROY*, *EDGEDESTROY*, or *NODESANDEDGESDESTROY*.
  - *NetworkModelModifyEvent*: The reason for the event can be on of *NAMEMODIFY*, *IDMODIFY* or *STRUCTUREMODIFY*. The *NAMEMODIFY* and *IDMODIFY* reflect changes to these fields. The *STRUCTUREMODIFY* means that the attached list of nodes and edges has changed, these are changes to their attributes or display properties.

ATTRIBUTE DEFINITION



The **attribute** package structure would contain the basic interface for an attribute and set of attributes.

- *AttributeModel*: interface to define the basic information for an attribute including id, name, namespace, parent namespace and data.
- *AttributeArrayModel*: interface to handle sets of attributes with the basic get, set and remove operations.

The **attribute events** package would contain events for create, destroy, modify and selection events (patterned similarly to network events).

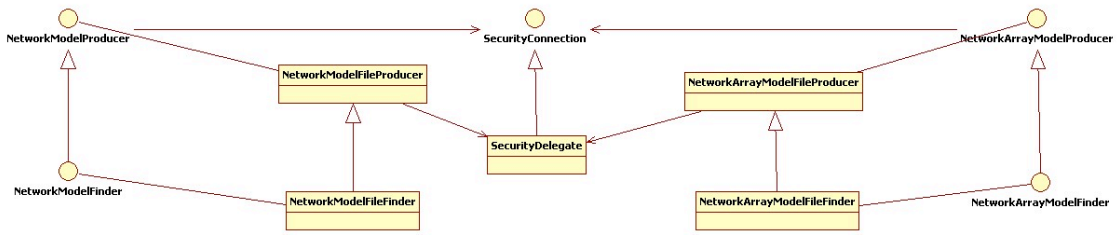
- The *AttributeModel* can fire the following events to the related listeners:
  - *AttributeModelCreateEvent*: Contains the identity of the new attribute model that has been created.
  - *AttributeModelDestroyEvent*: Contains the identity of the attribute model that has been destroyed.
  - *AttributeModelSelectionEvent*: Contains a selection object with a list of attribute models that have been selected. The reason is either *SELECTED* or *UNSELECTED.*
  - *AttributeModelModifyEvent*: The reason can be one of *NAMEMODIFY*, NAMESPACEMODIFY, DATAMODIFY. The *NAMEMODIFY* means that the string representation has changed, NAMESPACEMODIFY means that the namespace of the attribute or it's parent has been modified, DATAMODIFY means that the data contained by the attribute has been modified.

VIEW LAYER

View layer is dependent on the model. It provides visual representations of the model, and will include additional state information. Controller classes will not be used in the first instance, but will be introduced when/if a web based system is introduced. The planned views include:

- Macro network view:
  - The current network view that a user interacts with in the desktop version of Cytoscape
  - Web based network view for user interaction
- Micro network view:
  - A small view like the current "birds-eye-view" in the desktop
- Attribute table view
  - Table containing the attribute information for each object (network, node, edge) via a swing gui or an html table
- Attribute notes view
  - View of a single attribute visually attached to the object it is annotating, similar to Mac Stickies, a table or pop-up html
- Group (networks, nodes, edges, attributes) views
  - Groups of nodes or edges could be viewed with shapes around the grouped objects in a swing or html view, as separate sub-networks connected to a single parent node or groups of attribute notes (e.g. Mac Stickes)
- Portlets
  - View a network or set of networks (macro or micro) and associated attributes in separate portlets

IO LAYER



IO/Comms layer handles network creation from both remote and local locations. The system is designed to be extensible to a number of different scenarios (including web service, J2EE, database and file system based loading). A basic single login security model and location "property" are suggested.

Producers are used to access single Models (either *CyNetwork* or *CyNetworkArray*) and so require a unique id, whilst finders provide "searching" functionality. The searching specification can either be through a generic method or through specific method signatures. The security model takes login and location information (e.g. JNDI resource, file name). Initially only a file system based implementation is planned.

ADDITIONAL PACKAGES

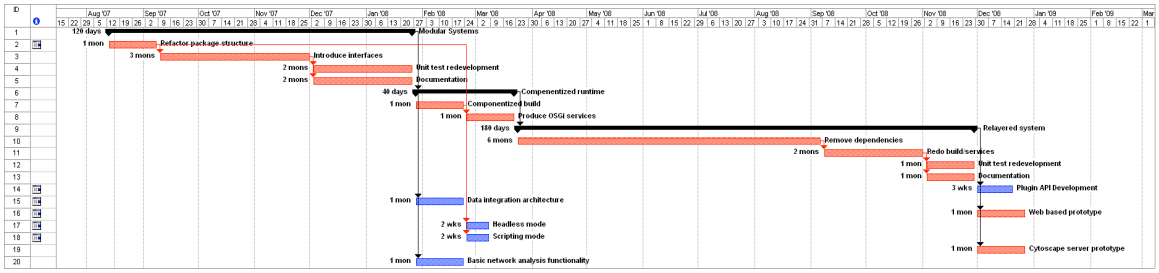
Application package will contain anything specific to the gui application (e.g swing for a desktop application).

- Dialogs required for user interaction with any part of the view
  - Menus/Menu bars
  - Various window panels
- Initialization classes to start the application and associated models/views.
- Process classes to handle the various tasks a gui application requires such as downloads, creating network views etc.
- Widgets for the discrete bits of the application that are specialized
  - Vizmapper (visual styles)
  - Actions to handle menu interaction
  - Layout classes to handle laying out the visualized networks
- Util classes

Common package will contain all the glue interfaces required as we separate out the layers. This would be a temporary package that will eventually be removed when the final layered system is complete.

PROJECT MANAGEMENT

PROJECT PLAN



TASKS AND MILESTONES:

1. ***Milestone/Deliverable 1: Completed modular systems.***
   a. Refactor the packages (estimated at 1 month): move the code into the new package structure. Starting with the model each package should compile in the

order demonstrated above. The model should have no dependencies outside itself. View, IO and Plugin depend on the model and the application will depend on Model, View and IO.
    b. Introduce interfaces (estimated at 3 months): use 'artificial' interfaces to break dependencies between classes in different packages. Starting with the model, write interfaces to allow other layers to access the model and remove all dependencies to classes outside the model. Once the model has no dependencies, work on other refactoring to ensure dependencies are correct and unidirectional.
    c. Unit test redevelopment and documentation (estimated at 2 months).

2. ***Milestone/Deliverable 2: Components based runtime***
    a. Produce componentized build (estimated at 1 month): build the system as a series of artifacts (using Maven or similar).
    b. Produce and register the relevant OSGi services (estimated at 1 month): Register each of the new components as services.
3. ***Milestone/Deliverable 3: First relayered system.***
    a. Remove dependencies (estimated at 6 months): Recode Cytoscape to reduce and repackage the artificial interfaces.
    b. Redo build and services/components (estimated at 2 months). First release of new componentized Cytoscape.
    c. Unit test redevelopment and documentation (estimated at 1 month).

PROJECT DEPENDENCIES

The following project depend on the successful completion of the milestones in this project.

- Plugin API - after the 3rd Milestone
    - includes OSGi wrapping
- Data integration architecture - after the 1st Milestone
    - Web services such as id mapping
    - See RFC 39
- Web based prototype - after 3rd Milestone
    - SVG or google toolkit UI
    - Contain some data tools for accessing public interaction data
- Headless - after 2nd Milestone
    - Command-line interface, requires good interface for commands that are available
- Scripting interface prototype - after 2nd Milestone
    - Ruby, Python, etc
    - Maybe use similar interface to the command line interface implemented in Headless mode
- Cytoscape server prototype - after 3rd Milestone
    - Include database of (some) interaction information
    - Lay web prototype over top for some gui functionality
- Basic network analysis functionality - after 1st Milestone
    - See RFC 41

ISSUES

There are a few issues that need to be resolved:

- Graph Layout manager: is there a requirement to have two views share the same layout instance. If so, then a layout manager/handler with graph update events is need (via a delegate/handler system attached to the model). Additionally should graph layout be through a separate model attached to the view, allowing for easier maintainability of graph layout occurring in a separate thread.

- Process Management: resource management is going to be needed if we wish to enforce "good behavior" of the system (and the plugins). If a process manager is needed, which will provide a basic state machine for thread based operations, then a process package should be introduced. This issue is related to the Plugin API RFC.
- Attribute change propagations should be channeled through a high level object to keep the number of (delegate) objects small. These could be through the *AttributeArrayModel* or through the *CyNetwork*.
- *NetworkArrayModelModifyEvent* could alternatively use a strongly typed system aligned with swing ListModel (including positional based insert based reasons).
- Setting up all network objects to be parts of lists (node lists, edge lists) to be able to create specific groups of these objects based on some parameter

2-B: CYTOSCAPE REWRITE

Sarah Killcoyne, John Boyle, Hector Rovira
October 29, 2007

PROPOSAL

Many Cytoscape clones are available that: utilize both web and server based technologies; use advanced information visualizations of graphs; and support a plugin architecture. While Cytoscape is still used preferentially, we can improve the functionality in a number of ways, including: clear scripting/macro capability; web based delivery; allowing for high throughput asynchronous message based analysis; server/distributed functionality; uniform access to data sources; and a componentized code base (e.g. to change the rendering code). In order to undertake this effectively there are two possible strategies, code refactor or code rewrite.

A refactor (See RFC 46) will take our current code base and reorganize it, giving us some opportunity to rework parts like the event handling but leaving us with essentially the same code we have now. If we consider a rewrite two strategies are available. Either we keep the current overriding architecture and develop a fat client or we move towards an alternative, thin client architecture. This would be a fundamental change in our code, but it would also move Cytoscape towards a modern, componentized architecture with the potential of being expanded easily into an enterprise system.

Either the fat or thin client architecture would allow us to separate the functionality cleanly for a more flexible and maintainable code base while easily adding more powerful functionality such as: a command layer for scripting and macros; a project based system; and full unit testing.

This proposal builds upon the following RFCs: (38) Code Layering, (40) Scripting in Cytoscape and (6) Cytoscape Headless Mode Operation, (46) Cytoscape Relayering.

USE CASES:

See Use Cases in RFC 46. Use cases specific to a rewrite:

- Faster to rewrite. In the 5 years that Cytoscape has been in development most of the issues relating to the core features will have been dealt with. Using what has been learned and potentially code currently in Cytoscape that has been well architected a rewrite will actually

be faster because the developers won't be tied to using old interfaces or spaghetti code that has been tacked on in the process of deprecating old code.

- Plugin rearchitecting can begin as soon as the basic model and IO layers are in place and continue throughout the process of writing view and application instead of having to wait (in the relayering RFC plugin rearchitecting cannot begin until the first fully relayered system is in place)
- A new project based system can be planned and implemented cleanly without requiring old code to be refactored.
- Tools that are integral and important to any basic network application such as search, filter, analysis and statistics can be added, drawing from work already done in core plugins.
- Full unit and deployment testing can be easily added to the build process as part of a rewrite making future feature addition simpler, cleaner and easily testable

BUILD DEPENDENCIES

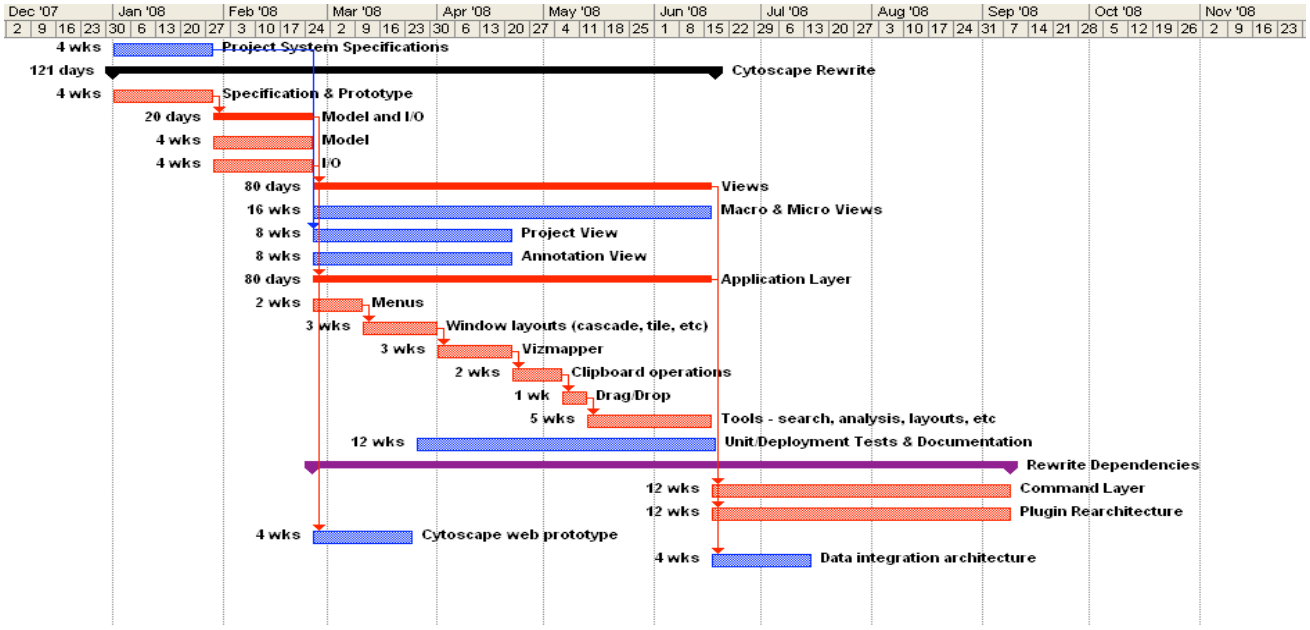
The proposal is to initially introduce four layers into the code: Model layer which contains the business logic; View layer containing all the graphical/presentation code; IO layer which controls readers/writers; and the Application layer containing all of the GUI systems such as menus, windowing and VizMapper.

FAT CLIENT ARCHITECTURE

PACKAGE STRUCTURE OVERVIEW

The package structure would be as described in RFC 46 with the exception that a Commons package for temporary interfaces would not be required.

PROJECT PLAN



TASKS & MILESTONES: ESTIMATE FOR 3.5 FTE'S 6 - 7 MONTHS

1. *Milestone 1 - Prototype*: FTE 1, 2, 3 - 4 weeks
    a. Specifications gathered and written
    b. Prototype written with package structure in place and basic views
2. *Milestone 2 - Model & I/O Package*: FTE 1, 2, 3 - 8 weeks
    a. Model package including model events
    b. I/O package
3. *Milestone 3 - View:* FTE 1, 2, 3 - 16 weeks
    a. Macro & Micro views take 8 - 16 weeks depending on decisions made about renderer (rewrite, use as is with interface updates or find 3rd party renderer) FTE 1, 3
    b. Project System view - 8 weeks. Depends on specification gathering and prototyping as this is a new feature. FTE 2
    c. Annotation view - 8 weeks. FTE 3
4. *Milestone 4 - Application:* FTE 1 - 16 weeks. Much of this should be easily portable from current Cytoscape application such as the VizMapper and current core plugins.
    a. Menus (new menu bar)
        - File
        - Edit
        - Window
        - Tools
        - Plugins
        - Help
    b. Tools
        - Search
        - Analysis
        - Statistics
        - Layouts
        - Editor
    c. Window layouts
        - Tiled, cascaded, etc
    d. VizMapper
    e. Clipboard operations
        - Copy, cut, paste
    f. Drag/Drop

OPTIONAL: 4TH FTE

- Unit/Deployment Tests and documentation, 12 - 16 weeks
- Command layer (allowing for scripting, macros, "headless" Cytoscape), 12 weeks

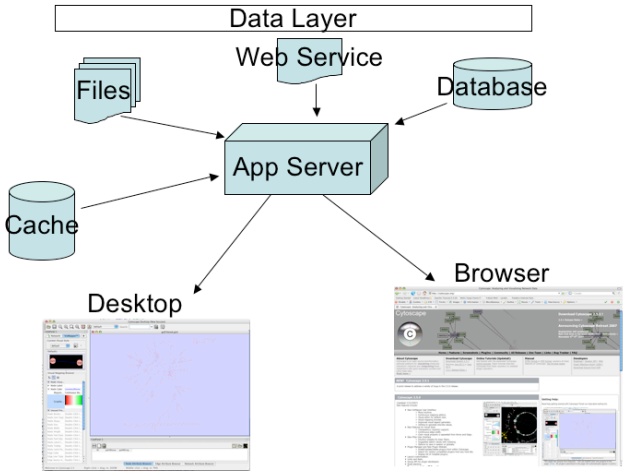
ISSUES

See Issues in RFC 46 - Cytoscape Relayering

THIN CLIENT ARCHITECTURE

PACKAGE STRUCTURE OVERVIEW

The layers of a thin client would be similar to those of the fat client architecture, but would be split



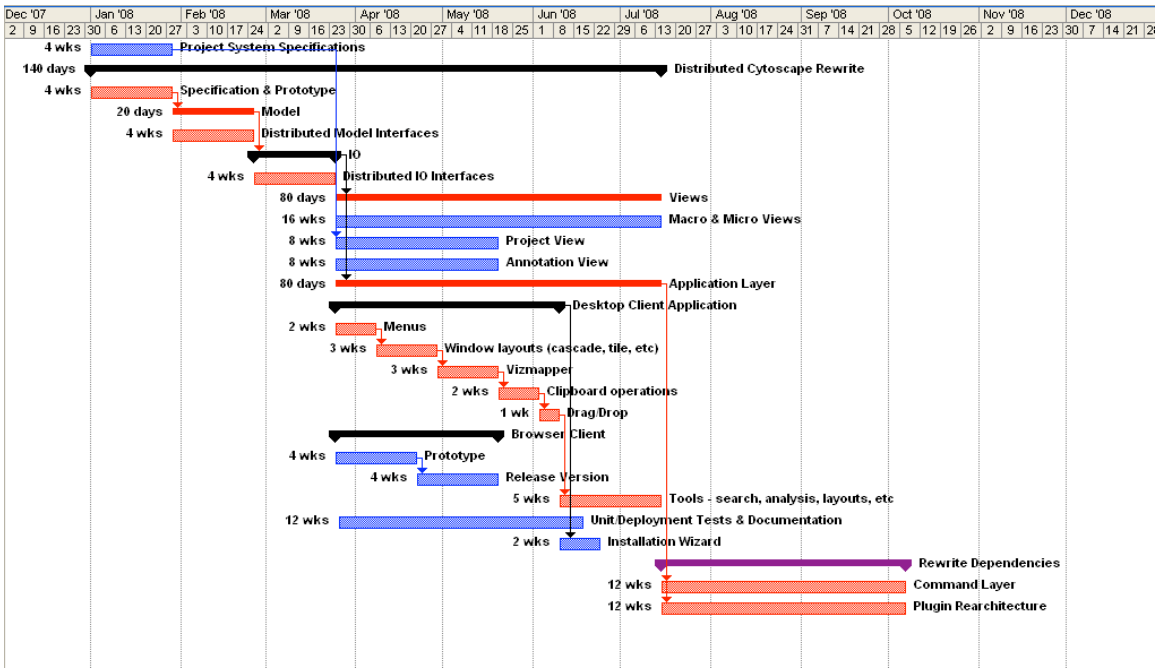
further.

The application server (Tomcat) would contain the API's for the *Model*, *IO* and perhaps the *View*. Plugins would be able to work as servlets (pushing their data to both the Desktop or Browser versions) or as part of the Desktop application in a jar file similar to their current installation.

This structure would take care a several current issues through the application server, including messaging, resource management and exception handling (negating the need for several current RFC's). It would also give us the ability to run Cytoscape as enterprise software for an entire institution using clusters for more computing power and private, local databases in addition to external data sources or continue to run as an individual desktop application similar to the Gene Pattern software developed by the Broad Institute at MIT.

PROJECT MANAGEMENT

PROJECT PLAN



TASKS & MILESTONES: ESTIMATE FOR 3.5 FTE'S 6 - 7 MONTHS

1. *Milestone 1 - Prototype*: FTE 1, 2, 3 - 4 weeks
    a. Specifications gathered and written
    b. Prototype written with package structure in place and basic views through Tomcat
2. Milestone 2 - Model & I/O Package: FTE 1, 2, 3 - 8 weeks
    a. Model package interface exposed through Tomcat
    b. I/O package interface exposed through Tomcat
3. *Milestone 3 - View:* FTE 1, 2, 3 - 16 weeks
    a. Macro & Micro views - 8 weeks FTE 1, 3
    b. Project System view - 8 weeks. Depends on specification gathering and prototyping as this is a new feature. FTE 2
    c. Annotation view - 8 weeks. FTE 3
4. *Milestone 4a - Desktop Application:* FTE 1 - 9 weeks. Much of this should be easily portable from current Cytoscape application such as the VizMapper and current core plugins.
    - Menus (new menu bar)
    - Tools
    - VizMapper
    - Clipboard operations
        - Copy, cut, paste
    - Drag/Drop
5. *Milestone 4b - Browser Application:* FTE 1, 3 - 8 weeks. This milestone is optional for the actual release version of Cytoscape via a distributed platform. It can be done at a later date.

a. Prototype - 4 weeks
b. Release version supporting basic Cytoscape functionality 4-8 weeks depending on technologies chosen (SVG, Ajax, Google Maps, etc)

6. *Milestone 5 - Installation Wizard*: FTE 2 - 2 weeks
   a. Bundle the application server (Tomcat) with the Cytoscape interfaces
   b. Create run/stop scripts for the Cytoscape server

OPTIONAL: 4TH FTE

- Unit/Deployment Tests and documentation, 12 - 16 weeks
- Command layer (allowing for scripting, macros, "headless" Cytoscape), 12 weeks

ISSUES

- See *Issues* in RFC 46 - Cytoscape Relayering
- There would be some infrastructure overhead and some changes in the development process

2-C: PLUGIN ARCHITECTURE REFACTOR

Sarah Killcoyne, John Boyle, Mike Smoot
August 24, 2007

PROPOSAL

Define the mechanism through which plugins interact with Cytoscape and can communicate with other plugins. This is closely related to the [:CytoscapeLayerRefactor: Layering RFC], as this describes the functionality we would want to expose. The mechanism need to be powerful enough to permit general operations applied to networks, maximizes reuse and allows for incorporation of new views, and addition of new import/export file types.

BACKGROUND

The rich functionality offered through the plethora of plugins is Cytoscape's greatest strength. As the software has evolved, and more plugins have been written, the lack of a defined plugin operation specification has become a problem. Currently plugins access core data structures directly so that any change in the core will break any number of plugins. Due to the tight coupling of the plugins to core code it is possible for a plugin to conflict with core processes and data structures, conflict with other plugins and even crash Cytoscape entirely. In particular, two problems are immediately apparent with the current architecture: the lack of resource management means that Cytoscape presents runtime robustness, performance and scalability problems; and the lack of backwards compatibility (through an established API or similar) means that plugins must be reimplemented between versions.

By creating a clean and functional set access mechanisms for each layer (see Layering RFC) plugins should be easily able to continue extending the functionality of Cytoscape while isolating them from changes made to core structures.

USE CASES

- Write a plugin once, continues to work in subsequent versions
- Resource management to prevent plugins from crashing each other or Cytoscape
- Standardize public interfaces

OVERVIEW

There are three central questions that need answering:

- What functionality should be exposed to a plugin?
- How should the functionality be exposed to a plugin?
- How do we manage the life cycle of a plugin?

Other topics for discussion:

- Can we provide a unit test framework that plugins an use to test against cytoscape?
- How will plugins communicate with one another?
- How will events be propagated to/from the core?
- Can we manage plugin resources or tasks centrally?

WHAT FUNCTIONALITY SHOULD BE EXPOSED TO A PLUGIN?

Based on the packages suggested in the layering RFC several external API's may be suggested. Only those that implement the interface that depends on the Model would be accessible to any implementation (such as scripting or headless mode) of Cytoscape that doesn't use any other layers.

1. *Model* - Basic commands available on any model.network object (networks, nodes, edges, attributes, groups):
    - Select
    - De-select
    - Modify
    - Delete
    - Create
2. *View* - Utilize any current views available such as layouts and create entirely novel views. Event/thread resources would need to be handled within the view package and parceled out to external code (plugins) separately.
    a. Create a pre-defined view on a model object
    b. Delete a view on a model object
    c. Modify a view on a model object
    d. Add/Create a new type of view (ex. molecular pathway view)
3. *IO*
    a. Add import file type
    b. Add export file type
    c. Add security information
4. *Application* - It may be useful to limit what/where things may be added to prevent confusion on the part of the user.
    a. Add a menu
    b. Add a panel (ex. CytoPanels)
    c. Access resources (e.g. threads, connections)

HOW SHOULD THE FUNCTIONALITY BE EXPOSED TO A PLUGIN?

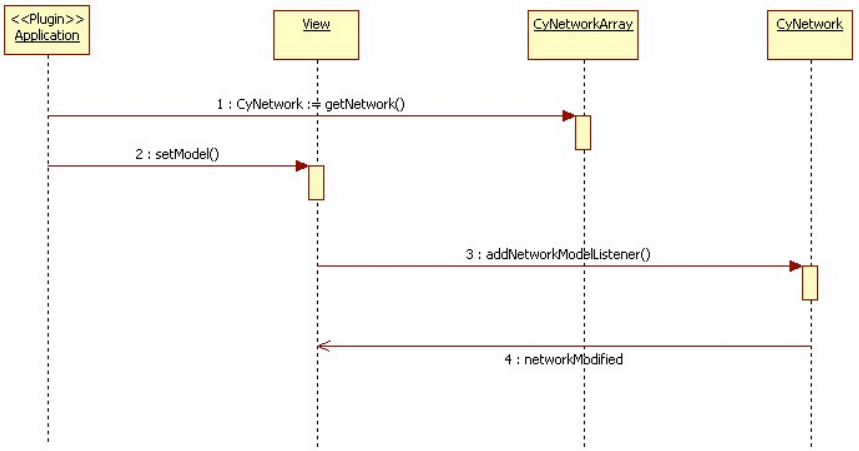
Opinions differ on how plugins should be managed, there are basically three different views:

1. *Interfaces*: a traditional approach would be to expose a set of interfaces which would remain stable between releases. The implementations of these interfaces could change, resulting in a simple (and understandable) "programming by contract" mechanism. While this is simple and elegant, problems with ensuring component compatibility will arise as requirements to change interface method signatures occur.
2. *Component*: allow for components to discover each other using a high level management system (e.g. OSGi). This does help to ensure component compatibility, but there are questions as to how the "plugin framework" (e.g. cytoscape core) can be componentized.
3. *Aspects*: annotating methods presents a modern mechanism for allowing support for plugins, and has been used successfully used in a number of containers. It can be used to overcome limitations associated with interface based programming (e.g. by allowing for dependency injection). While it will allow for the evolution of interfaces, there are problem as POJOs are not suitable for all plugins.

The correct solution can, and probably should, draw in all three views to allow for the development of a plugin system which is useful and maintainable.

INTERFACES

This is the simplest mechanism is to expose interfaces and finders for the associated instances. The diagram below illustrates how to find a specific model and listen for changes to the model.



Although interfaces provide for a degree of backwards compatibility, they are limited to a freeze on method signatures. Small incremental changes to interfaces can possibly be absorbed through the use of dual class loaders, with method call delegation being used.

The interfaces that we can expose are described in the Layered RFC (46).

COMPONENTS

A large number of component based frameworks and architectures exist, and range from simple patterns (e.g. JavaBeans) to dynamic discovery systems (e.g. OSGi). We can encourage plugin developers to follow/adopt a component based architecture enabling better support for both the discovery of plugins (runtime or otherwise) and provide a system for "auto updating".

The core components within Cytoscape could also be made available through OSGi, and inter-plugin communication could also be supported. This would require:

- Using a taxonomy to describe plugins (extension of what occurs currently for the plugin manager)
- Creation of a service registry to define where to register/find a plugin

Components do not remove the problems with backwards compatibility, but do provide for:

- Better organization of code
- Runtime discovery of plugins and inter-plugin communications
- Better reliability, as the different versions of plugins and core code can be made available.

The core components that we can expose as services are:

- NetworkArrayModel component (in the end, there can be only one), which can be used to find the list of currently loaded models of graphs. The user can then listen for events for modifications.
- Application component to allow for menubar merging and similar operations
- Each of the views, these would each be a separate component with a unique name. A hierarchical taxonomy based on tasks (e.g. macro view) and domain ("single network view") could also be used to ensure the system is both transportable to other platforms and is extensible.

ASPECTS

AOP offers a means to provide for both runtime and compile time injection of code. It can be used to dramatically simply frameworks integration (e.g. EJB 2.x versus EJB 3), and allows for code to be easily transportable (e.g. POJOs). Unfortunately it is relatively easy to overuse aspects. Such overuse results in too much functionality being made available through a somewhat arbitrary list of annotations. By carefully considering if/when we wish to use this powerful, albeit confusing, programming construct we can alleviate some of the problems associated with plugin maintenance.

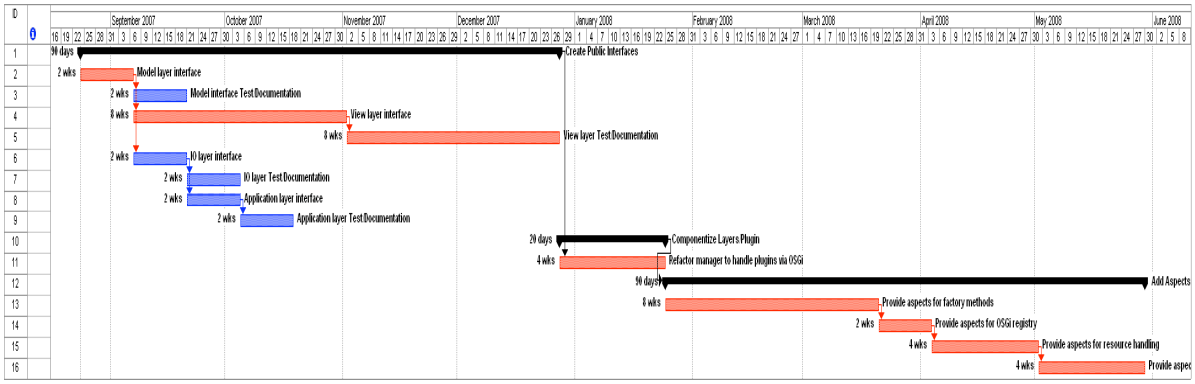
We would suggest encouraging the use of aspects to support:

1. The registration and naming of plugins e.g. http://static.springframework.org/maven2/spring-osgi
2. Replace any factory methods that are required with attributes (e.g. inversion of control)
3. Support multiple delivery mechanisms for plugin functionality (e.g. WebService tags)
4. To support plugin developers "nice" (thread management) use of resources (e.g. "link during configuration not compliation")

Aspects can be made available to developers, and can be backed up by interfaces. In this way we can encourage, but not enforce, their usage.

PROJECT MANAGMENT

PROJECT PLAN



TASKS AND MILESTONES: ALL TIME ESTIMATES BASED ON 1 FTE.

1. *Milestone 1: Develop Interfaces*
    a. Develop interface to the model layer (estimated at 2 weeks) to handle the basic commands outlined above. Should be compilable with only the model as a dependency.
    b. Unit test/documentation (estimated at 2 weeks) for model interface
    c. Develop interface to the view layer (estimated at 8 weeks). This includes resource handling for parceling out events to plugins as well as the create/delete/modify/add commands above. Should be compilable with the model and view as dependencies.
    d. Unit test/documentation (estimated at 8 weeks) for view interface
    e. Develop interface to the IO layer (estimated at 2 weeks) to handle adding import/export file types.
    f. Unit test/documentation (estimated at 2 weeks) for io interface
    g. Develop interface to the application later (estimated at 1 week) to handle adding menus and actions to the menu bar.
    h. Unit test/documentation (estimated at 1 week) for application interface
2. *Milestone 2: Componentize Layers/Plugin*
    a. Refactor manager to handle plugins via OSGi. This includes creating/managing the registry (estimated at 8 weeks)
3. *Milestone 3: Add Aspects*
    a. Provide aspects for factory methods (estimated at 8 weeks)
    b. Provide aspects for OSGi registration (estimated at 2 weeks)
    c. Provide aspects for resources (estimated at 4 weeks)
    d. Provide aspects for application and view layer (estimated at 4 weeks)

ISSUES

1. Do we want plugins to communicate with each other?
2. Do we want the core system available as separate components. If so how should we divide the system up (e.g. by layer, by domain or by both)
3. To what level do we wish to support web delivery – do we want cytoscape plugins to readily be available a tomcat webapp, do we want alternative (e.g. portlet) views?
4. An audit of all the major plugins needs to be undertaken to ensure that the required functionality is available.

5. Need to establish what aid we wish to give to plugin developers (e.g. are all we going to provide is a cytoscape_plugin.jar)?

2-D: EVENT HANDLING

Sarah Killcoyne, Allan Kuchinsky, Mike Smoot
October 24, 2007

PROPOSAL

The current event handling in Cytoscape has become too convoluted even for the core developers at times. This makes consistent usage in core code difficult, and makes it even harder for plugin developers to understand and access events they may have an interest in. As part of the rearchitecting of Cytoscape we can create a coherent event system and define the events that plugins can have access to, as well as trying to run those events in a manner that helps prevent plugins from tripping over each other when listening for the same events. This will also allow us to create a command layer for macros and scripting fairly easily.

USE CASES

Three main cases (examples, not comprehensive lists):

1. Core code event usage
    - Views being created in response to network creation
    - Views modifying in response to network modification
    - VizMapper modifying a node visual in response to attribute creation/modification
2. Plugin event usage
    - Adding an option to a popup on a right-click of a node
    - Expanding or navigating subnetwork based on a double-click of a node
    - Creating/applying a particular visual style in response to a network creation
3. Command layer
    - macros
    - scripting

IMPLEMENTATION PLAN

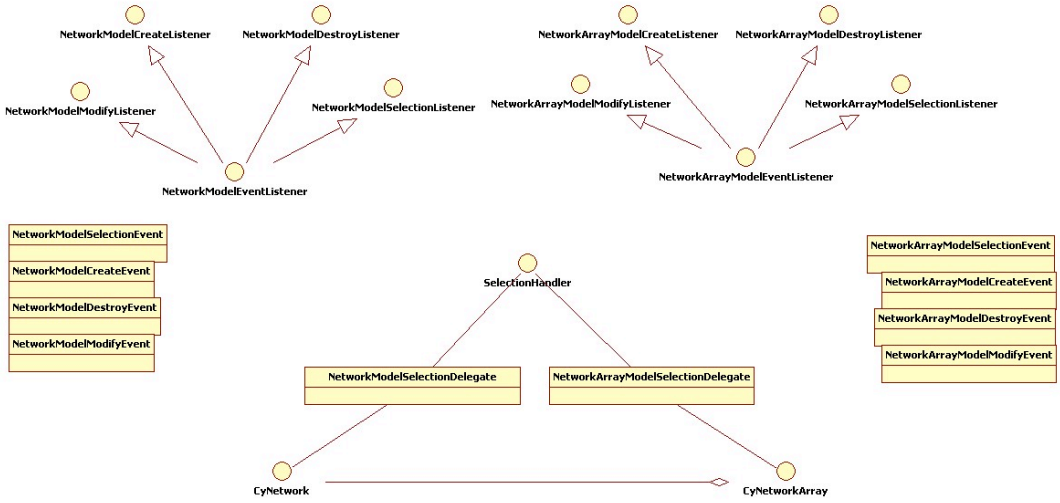
This plan is a sub-part of the relayering process (RFC 46 - Cytoscape Relayering). Based on that there are two or three areas for events to be spawned:

1. Model - creation/modification/deletion of a model object such as a network or attribute.
2. Application - actions (used by menus or macros), mouse events, keyboard shortcut events
3. Views - depending on how the view and the application are separated there may be space for events about the creation/modification/deletion of a particular view.

All of these events may be integration points for plugins. At the application and view levels an event queue for plugins would be added. Plugins would register for events and be added to the queue then run in whatever order they were registered. The model layer could also use a queue, but it would be less necessary since these events are not gui related.

MODEL EVENTS PACKAGE STRUCTURE:

From RFC - 46



APPLICATION/VIEW EVENTS PACKAGE STRUCTURE

This is greatly dependent on how the Application and View packages are separated. However, Views can have an event structure similar to the model (create/modify/select/destroy events) and the Application events will be the basic swing gui events.

Potential integration points for plugins in these packages include:

- Mouse clicks
- Keyboard actions
- Views created/modified/selected/destroyed
- Popup menus on nodes/edges/attributes
- Drag/drop
- Cut/paste
- others...

PROJECT MANAGEMENT

PROJECT TIMELINE

The creation of an event handling system would be part of a re-architecting. As part of the relayering of Cytoscape (RFC 46) the event handling can be restructured during the refactor of packages (Milestone 1). This may add some time (est. 8 weeks) to the initial refactor in order to untangle the current event system and add a queue system.

TASKS & MILESTONES

See the Tasks & Milestones section of RFC 46.

ISSUES

- ***Allan Kuchinsky***: Mouse event handling has been a particular source of conflict. It would be useful if there were a better way to arbitrate between listeners of a mouse event. In some cases plugin developers attempt to isolate specific actions in their plugins by defining modifiers for mouse events, e.g. the Cytoscape editor uses control-click as a shortcut for adding nodes and/or edges. Modifiers can't be relied upon for arbitration, though. For example, a plugin that associates an action with the modifiers control- and alt- on mousedown will not be isolated from the editor addition of nodes and edges because the editor only checks for control- modifier, so it will catch control-, alt-click events as well. For the application code to be isolated, the Cytoscape editor would have to check that control-modifier is on and alt-modifier is not on. Some sort of conventional scheme for masking event modifiers might make sense.

- ***Mike Smoot***: Even though this is going to get complicated, I think we should strive to keep things as simple as possible. I've been thinking of something along the lines of a single CytoscapeEvent class which is typed using an enum, similar to what we're doing with visual properties. CytoscapeEvent would have a simple interface, similar to PropertyChange event. Users would register a listener for a specific event type and then handle as they see fit. We could get creative with this if we want (i.e. each event type could return an object of a specific type) to provide more clearly defined functionality. The separation of different event types into classes is what GINY tried to do initially and (I feel) has contributed to the current confusion because each event has it's own event, it's own listener, and own set of methods. In addition, much of the event handling is channeled through methods in listener interfaces, which makes the interface inflexible and hard to change. By defining one or two interfaces that can be extended and adapted for different needs, we should avoid some of this difficulty. I like the separation of Application/View events from Model events. My only question is whether we want the Application/View events to include things like mouse click events or menu selection events. It seems to me that if plugins are limited to "adding a panel" to a gui, then that panel would be in charge of its own internal click events and any other interaction with the Application/View would be through defined events. I'm thinking along the lines of how ContextMenus now work. Instead of users listening for right clicks on nodes, they would listen for a CytoscapeEvent with type NodeSelection. Or something along those lines.

- ***Scooter Morris***: I really like the idea of "keeping it simple", but I'm wondering if there might be value in having three event classes: CyViewEvent, CyModelEvent, and CyAppEvent or something like that. There sort of seems to be different levels of granularity between the three which might benefit from some different semantics. App events should be moderately rare, Model events could be pretty common and there might be some benefit of handling these differently (as lists?). I see some efficiencies in batching model events together. View events might fall somewhere in between, but I suspect that they would be closer to the app semantics, so maybe they could be merged.

2-E: RESOURCE (TASK) MANAGEMENT

Scooter Morris
October 17, 2007

PROPOSAL

This proposal, part of the Cytoscape 3.0 project, is to redesign and refactor the task management and monitoring capabilities within cytoscape to provide a more robust task management and task monitoring system.

BACKGROUND

Cytoscape's current Task, Haltable, and TaskMonitor interfaces have provided useful hooks to monitor, and to a limited extent manage, Cytoscape tasks. Using Cytoscape's task system handles much of the detail of setting up the progress bar, etc. Unfortunately, for a number of reasons, using Cytoscape's task system can lead to significant complexity since it is currently designed to monitor/manage only one task. In addition, many of the current tasks are not coded to be canceled, and with the deprecation of Thread.stop code that is running within a task needs to be coded to specifically monitor a volatile variable to signal task cancellation or catch InterruptedException.

USE CASES

1. When reading an XGMML formatted file from a web site, there are several tasks that much be accomplished. First, the exchange with the website needs to be handled, this would include the initial query, plus any responses from the query. In some cases, the initial response might take an indeterminate amount of time. Second, the stream needs to be read from the website and parsed, and finally, the network view needs to be created, which includes calling the reader to instantiate all of the visual attributes of the network. There are at least two separate tasks that could be created to handle this use case: an HTTP reader task, which includes the initial handshake; and the parser task, which might include the network view creation. Currently, this is handled through a complicated process of handing around (possibly null) TaskMonitor objects and there is no easy way to have more than one task monitored simultaneously.
2. When reading in sessions, we handle several different pieces of information, some of which are independent. Currently, these are all handled sequentially. If we were to parallelize this process, we would need some mechanism to monitor and manage the overall progress.
3. Some plugins take advantage of existing Cytoscape services and combine them in important ways to achieve a desired result. For example, a plugin that downloads information from a website, performs some filtering or analysis on the downloaded network, then performs a layout and/or application of a visual style would currently need to manage each of these tasks independently, some of which might not be easily canceled.

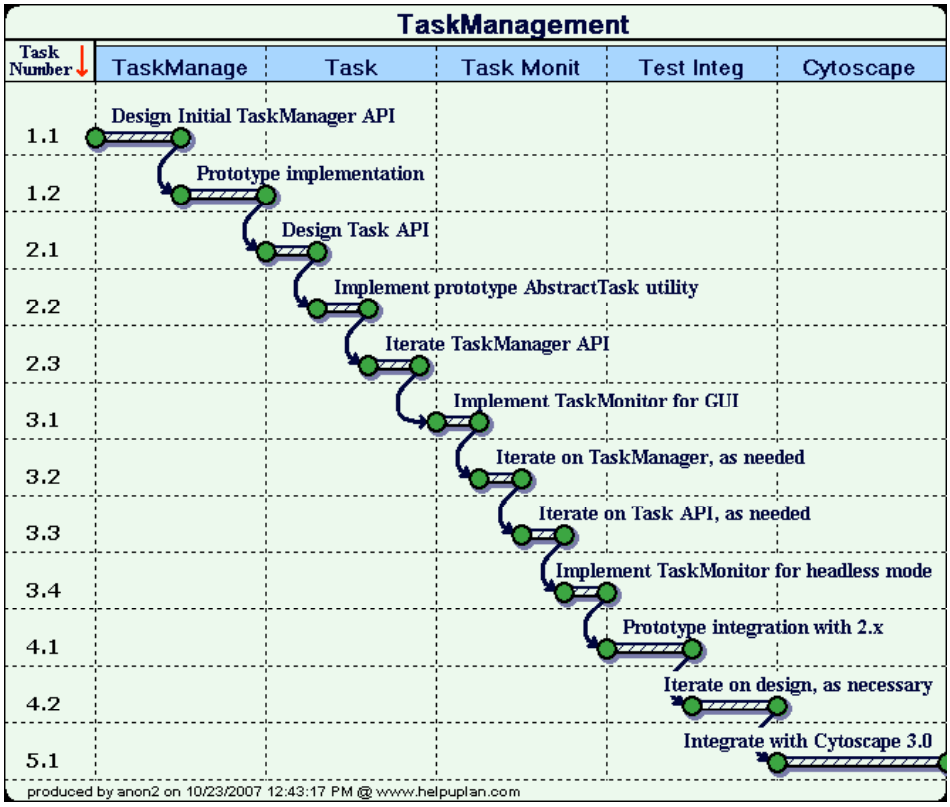
IMPLEMENTATION PLAN

The basic idea is to replace the cytoscape.task package with a more robust mechanism that handles the management of multiple tasks, task groups (multiple tasks that should use a single progress monitor), headless management of tasks, and user-initiated task cancellation. Some of this will depend heavily on the Cytoscape3.0 logging/error handling capabilities. In order to support good separation of concerns, a Cytoscape Task must be independent of the

TaskMonitor. There will need to be a TaskManager object that will manage all Cytoscape tasks. When a Task is initiated by a call to create a new cytoscape.task.Task (or perhaps the run method of a cytoscape.task.Task object?) it will register itself with the TaskManager, along with a label or task group. The TaskManager will be responsible to providing task monitoring services (if required) as well as rudimentary task logging services (using the 3.0 logging/error handling capabilities).

PROJECT MANAGEMENT

PROJECT TIMELINE



TASKS & MILESTONES

1. *Milestone 1: TaskManager*
    a. Task 1: Design Initial TaskManager API
    b. Task 2: Prototype
2. *Milestone 2: Task*
    a. Task 1: Design Task API
    b. Task 2: Implement prototype AbstractTask utility class
    c. Task 3: Iterate TaskManager API and implementation, as needed
3. *Milestone 3: Task Monitoring*
    a. Task 1: Implement TaskMonitor for GUI
    b. Task 2: Iterate on TaskManager API, as needed
    c. Task 3: Iterate on Task API, as needed
    d. Task 4: Implement TaskMonitor for headless mode
4. *Milestone 4: Cytoscape 2.7 Test Integration*
    a. Task 1: Prototype integration with Cytoscape 2.7 (not for release)

b. Task 2: Iterate on design, as necessary

5. *Milestone 4: Cytoscape 3.0 Integration*
    a. Task 1: Hook task capabilities into Cytoscape logging and error handling features

ISSUES

2-F: CYTOSCAPE PROJECTS

Scooter Morris, Sarah Killcoyne
October 23, 2007

PROPOSAL

Extend the concept of sessions to a more flexible project based system where all of the data involved in a given project is transparent and accessible for altering or copying to another project. This would help biologists to work in the way they are used to, where one project may build off of another and use many of the same properties or data files. For example, a Project would "remember" that a network (saved as an XGMML file) was created by reading in a SIF file, and associating three node attribute files and two edge attribute files with it. The goal would be data provenance, not necessarily to be able to recreate the initial data sets. Moving forward, Projects could include logs (Cytoscape 3.0 logging), user created scripts/macros, workflows, etc. While the Projects panel might list all available projects, only one project would be opened at a time. As a result, a project could replace a Cytoscape Session as the main mechanism for saving state.

BACKGROUND

Scientists are accustomed to thinking about their work in terms of projects where all of the various types of data and analysis they have done are stored together, but separately accessible for use in other work. Many pieces of open and proprietary software used by scientists use this concept now to help organize and share work (such as CPAS and SBEAMS). Cytoscape does nothing to help users track their data provenance. Where did these attributes come from? How was this network created? What steps (including what data sources used) were performed to expand the network? All of these are legitimate questions that Cytoscape provides no means of assisting users to remember.

USE CASES

- Tracking what data and sources were used to create or expand a network
- Easily copying a visual style from one Project to another allowing users to continue using their work (rather than creating it de novo each time)
- Sharing all of the data, properties and attributes easily
- Saving the full state of Cytoscape in a given project including:
    - Properties
    - Data
    - Plugins/Themes loaded or used
    - Macros/scripts
    - Others?

IMPLEMENTATION PLAN

There are two major parts to implementing a project system in Cytoscape. Both will be simplified through the proposed refactor of Cytoscape.

1. EXPAND THE DEFINITION OF A SESSION TO INCLUDE DATA PROVENANCE INFORMATION

In a relayered system the hooks into loaders in the IO layer can be added to track the data imported into a project from any source. Hooks added to the Application layer would allow for the association of scripts, macros or workflows with a particular project as well.

2. VISUAL REPRESENTATION OF DATA PROVENANCE

This visual representation can be dealt with in the View and Application layers. The views on a network and information about the data in that network can be provided to the user in a format similar to that of the Eclipse IDE where projects contain trees of information that can include the type, date and path of any particular piece of data within the tree. This sort of system would make it easy to add new features to a project such as a log file that could function as a sort of laboratory notebook, recording the exectution and results of Cytoscape actions (including plugins) and notes recorded by the user.

PRIOR TO RELAYERING

Hooks into the current network and attribute loaders can be added. The current network view panel can be replaced by a Project panel that can initially list the network and attributes loaded and potentially the visual style currently applied. This panel can also start by listing the network, attributes and visual style. This particular work is not necessary to a 2.x release but could help prepare users for the switch.

PROJECT MANAGEMENT

PROJECT TIMELINE

This work will depend on the relayering of Cytoscape and can be started during the refactoring of packages. See RFC 46.

TASKS & MILESTONES: 3.5 MONTHS, 1 FTE

1. *Milestone 1: Create Project Model (est. 4 weeks)*
2. *Milestone 2: Track Loaded Data (est. 6 weeks)*
    a. Add hooks into the IO package to track loading of data
3. *Milestone 3: Project View (est. 4 weeks)*
    a. Create view of project that allows registration by various sources (for easy extension for actions, macros, logging etc at a later point) for the data loaded in IO
4. *Milestone 3: Project Panel (est. 4 weeks)*
    a. Create a panel in the Application package to display the Project View

ISSUES

2-G: WEB FRONT END (PROOF OF PRINCIPLE)

Sarah Killcoyne, Allan Kuchinsky
October 24, 2007

PROPOSAL

Cytoscape offers a lot of functionality for a desktop user, but more and more scientific work and collaboration is done via the web. Many well used applications also offer both a web and desktop version. In order to allow Cytoscape users to use the web for their work, a basic web front end can be add to Cytoscape after relayering of the core has been finished to provide simple access to the *Model* and *IO* parts of the core.

BACKGROUND

This is a proof of principle project that will be done after the relayering of Cytoscape (RFC 46). Some aspects of the design of the project, e.g. learning about and deciding upon a choice of alternative web-based graphical rendering packages (e.g. AJAX, GWT, Flash), can be done concurrently with the relayering of Cytoscape.

USE CASES:

- Institution wide server version of Cytoscape, encouraging wider usage
- Facilitate collaboration through simple sharing of networks and data
- Basic network functions available to new users
- Simplify collaborator usage, such as http://t1dbase.org or http://www.reactome.org

SCENARIO: COLLABORATION VIA SIMPLE SHARING OF NETWORKS AND DATA.

You and I are molecular biologists investigating cardiovascular disease. We work in different labs in different institutions. I have been using Cytoscape for two years, as a standalone application. You have never heard of Cytoscape and generally try to avoid installing and learning complicated new software. You use the web daily for literature searches, email. We both use microarrays and statistical data analysis programs. I have done some time course studies using microarrays, have built a biological network via literature search using a list of most differentially regulated genes in the microarray data, and have identified a small number of genes which appear to be acting as switches for regions of activity -- they are highly connected in the network and tend to be up-regulated when their neighbors in the network are down-regulated, and vice versa. This hypothesis is consistent with what I observe when I overlay my data on the network. When my putative targets are up-regulated, their neighbors are down-regulated, and vice versa. I want to corroborate my hypothesis by sending my biological network with you and having you perform the same kinds of data overlay using microarray data that you have generated independently. I email you a URL, which, when you traverse it, downloads my Cytoscape network and displays it in your web browser. You select a microarray data file in your Finder window and drag/drop it onto the web browser page. The browser client reads in the file, parses it, and presents a spreadsheet (or heatmap) view of the data. You step through the spreadsheet (or heatmap), selecting a column at a time and observing how the node colors change. You notice a similar pattern except in the case of one group of samples which have been drawn from patients who had been treated with a certain drug. You feel that this is an anomaly worth further study and tell me so in your email

response to me. You also take a closer look at one gene which you think is particularly interesting. You perform a control-click mouse gesture and see a context menu that lists the names of a number of database resources. You select one of the menu items and a browser window is brought up with detailed information about the gene of interest. In the meantime, I run another literature search, this time adding the name of the drug into the search context, which generates a new network, which has noticeable differences from the first one. I overlay my microarray data on this new network and we do another couple of iterations of data visualization and network inference in this manner.

IMPLEMENTATION PLAN

The view and application layers of Cytoscape can be swapped out to provide a web front end through use of web technologies including Tomcat, SVG and/or the Google web toolkit.

The scenario above requires the following functionality in Cytoscape browser client:

- import a network by retrieving it from a server, given a URL
- perform some simple graph layout algorithm to generate node coordinate positions
- display the network -- read only, no editing needed. We also assume relatively small networks initially.
- lookup detailed information about nodes of interest -- via linkout or equivalent technology.
- import attributes into the browser client interest and map the attributes to nodes in the network.
- overlay data using a fixed visual style with fixed mappings and a simple spreadsheet-like interface. Selecting anywhere in a column of the spreadsheet causes the nodes in the network to light up according to their data values in the selected column.

A first implementation of this front end would be limited to the minimum software required to support the functionality of the scenario above. Additional functionality could be phased in to support capabilities such as:

- user definition of visual properties using the vizmapper
- simple filtering
- editing networks
- panning/zooming
- small subset of layout algorithms
- support for decent rendering speed on larger networks
- analysis and visualization plugins available as web services. These would probably provide their own browser-based UIs, but a Cytoscape web front end could provide some common services, such as menus.

The goal is *not* to duplicate the full functionality of Cytoscape in a web front end, but rather to provide a subset of Cytoscape that is sufficient to support a conventional workflow for a non-computer-oriented life user.

PACKAGE STRUCTURE OVERVIEW

It is estimated that a basic implementation could be provided in 6-8 weeks after relayering has been done. After a prototype has been worked out a release version could be done in 12-16

weeks time. An actual timeline can be worked when the relayering gets to a point that the correct hooks are available for Views/Applications to be added.

PROJECT MANAGEMENT

PROJECT PLAN

TASKS & MILESTONES

- Prototype web front end (est 6-8 weeks)
- Write new RFC describing release version (est 1 week)
- Release version (est 12-16 weeks)

ISSUES

1. Are we assuming that this is an infrequent user?
2. Are we assuming that this user always works in collaboration with a user who has full version of Cytoscape?
3. What are the further implications of 1 and 2 above?

2-H: DOMAIN SPECIFIC MODELS: CYTOSCAPE "THEMES"

Alex Pico
October 24, 2007

PROPOSAL

While it is important that the semantics in the core of Cytoscape remain domain-agnostic, we also recognize the need to support use cases and plugins that are very domain-specific. We propose the concept of Cytoscape Themes to provide plugin developers and users with flavors of the Cytoscape model designed with specific research domains in mind. By providing a few basic overlays and defined standards we can greatly facilitate plugin development and improve user experience.

This proposal touches on a number of areas including domain-specific semantics, data integration and user interfaces. We propose to do the following:

- Overlay domain-specific syntax onto the data model (a.k.a syntactic sugar), while maintaining the agnostic core model. Overlaying domain-specific syntax onto the data model will clarify the model and help direct plugin developers who are implementing domain-specific functionality (e.g., microarray analysis).

- Define annotation standards per domain to facilitate the mapping of external data sources to a standard set of attributes. This will allow all plugin developers within a particular domain to take advantage of commonly shared attributes, knowing where to map data to and where to find data.

- Make changes to the view to reveal the appropriate semantics to the user in the form of interfaces, menus, panels, labels, etc. Thus, the benefits of the "sweetened" syntax

should trickle down to the end user by making it easier to load and map datasets and interpret domain-specific visualizations.

The first step, however, is to re-layer the core Cytoscape code and settle the API (see RFC 46 and RFC 47). We should keep Cytoscape Themes in mind as one of many motivations for performing these crucial tasks.

USE CASES

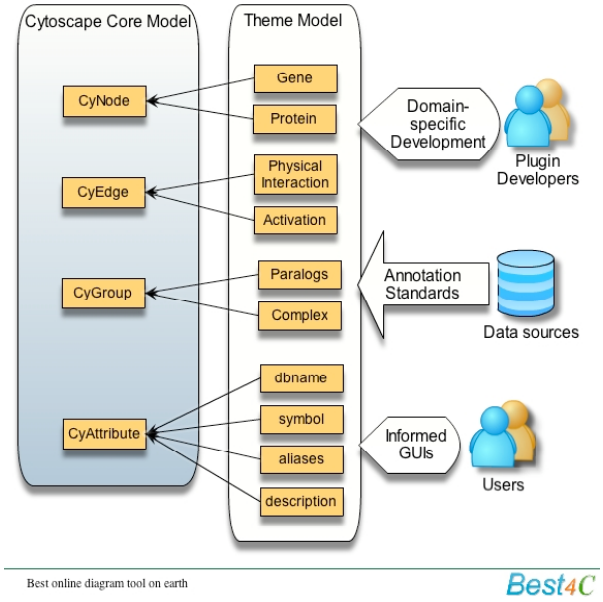
- Biological Pathways: semantics could be extracted directly from BioPAX, this would support pathway analysis of microarray or proteomics data for example.

IMPLEMENTATION PLAN

Once the core Cytoscape code has been re-layered and the API is settled, we can develop a set of modular plugins to define a given theme (see basic biological example below). These plugins, and their respective APIs, can be used by developers and, in turn, be bundled with analysis plugins to facilitate installation and use by the end-user (see Cytoscape Plugin Bundles RFC).

The following plugins would, together, support the basic biological semantics for pathway analysis of microarray data, for example.

1. Overlay basic biological syntax onto the core data model. Here are a few examples:
    - 'Gene' extends 'CyNode'
    - 'Protein' extends 'CyNode'
    - 'Paralogs' extends 'CyGroup'
    - 'Complex' extends 'CyGroup'
2. Define annotation standards. For example, ID mapping from various data sources:
    - database name (standardized like http://www.geneontology.org/cgi-bin/xrefs.cgi)
    - identifier
    - symbol
    - full name
    - aliases
    - genomic location



Variations on these plugins could be mixed, matched and bundled to support various domain-specific tasks.

PROJECT MANAGEMENT

PROJECT TIMELINE

Not yet determined.

ISSUES