Cytoscape 3.3 Developers Tutorial

John "Scooter" Morris, Ph.D., UCSF

Outline

- Introductions & Setup
- "Variations on a theme – Hello World"
  - Step 1: Cytoscape and OSGi
  - Step 2: Cytoscape network model
  - Step 3: Cytoscape table model
  - Step 4: Cytoscape visual model
  - Step 5: Cytoscape user interface mechanisms
  - Step 6: Events
  - Step 7: Commands
- API tour
- Discussion: Best Practices

Introductions

- John “Scooter” Morris
  - Currently
    - Adjunct Assistant Professor, Pharmaceutical Chemistry
    - Director, NIH Resource for Biocomputing, Visualization, and Informatics (RBVI) @ UCSF
    - Roving Engineer, NIH National Resource for Network Biology (NRNB)
  - 1985-2004
    - Distinguished Systems Architect: Genentech, Inc.
  - Cytoscape core team since 2006
  - Author of several Cytoscape plugins
    - SFLDLoader, *structureViz, clusterMaker, chemViz*, metanodePlugin, groupTool, commandTool, bioCycPlugin

Introductions

- What do you hope to accomplish today?

Caveats

- I'm assuming you're all competent Java programmers
- Can't cover everything
  - You would be overwhelmed even if we tried
- We're going to cover Cytoscape 3.3 Bundled App API
  - Simple apps are different in many ways
- We'll be building on things as we go
  - Stop me and ask questions

Setup – Shell

- Download sample files from:

http://www.cgl.ucsf.edu/home/scooter/Cytoscape3DevTut/setup.zip

- Unpack

Setup – Eclipse

- Install Eclipse
  - Also want:
    - m2e
- Download sample file
- Unpack
- In Eclipse:
  - File→Import→Maven→Existing Maven Projects
  - Navigate to the HelloWorld directory from the download above

Debugging with Eclipse

Remote Execution for Debugging

- Run Cytoscape with debug as the first command line argument

In Windows

```
cytoscape.bat debug
```

In Linux/Mac

```
./cytoscape.sh debug
```

Make sure you can see the following message in the terminal:

Listening for transport dt_socket at address: 12345

NOTE: must be in the Cytoscape directory

Debugging with Eclipse

Run Debugger From Eclipse

- Click Run and select Debug Configurations...
- Select Remote Java Applications and create new configuration
- Select/Enter the following and press Apply:
  - Name: Cytoscape3
  - Project: myproject (your project name may be different; so "browse" to find the one you are about to debug)
  - Connection Type: Standard (Socket Attach)
  - Host: localhost
  - Port: 12345 (or the port specified in the cytoscape.sh file)
- Press Debug. This starts up Cytoscape 3. After 20-30 seconds, you will see Cytoscape Desktop.
- Set a breakpoint in Eclipse
- Switch Perspective to Debug mode: Window → Perspective → Debug
- Load the bundle into your OSGI container and run

Step 1: Cytoscape & OSGi

- Cytoscape 3 design goals
  - Scalability
  - Performance
  - Stability
    - Application stability
    - API stability
    - Modularity
      - Enforced by OSGi

OSGi

- Wikipedia:

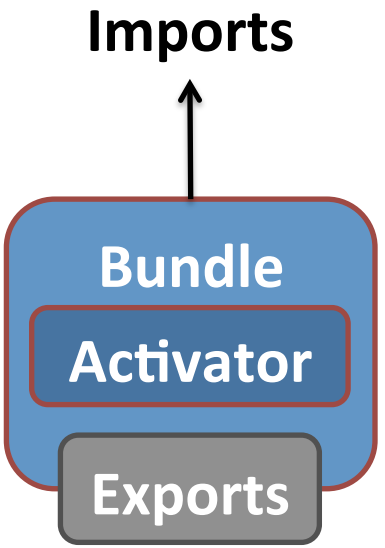
  *The **OSGi framework** is a module system and service platform for the Java programming language that implements a complete and dynamic component model, something that does not exist in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/ classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.*

OSGi

- Service-oriented
- A “bundle” is the unit of access
  - Bundles can be started and stopped independently
- Bundles implement services
  - Can be registered and unregistered
  - Generally, inter-bundle access is through a service
- Enforced separation of API and Implementation
  - Rules are that you can depend on API bundles, but not implementation bundles

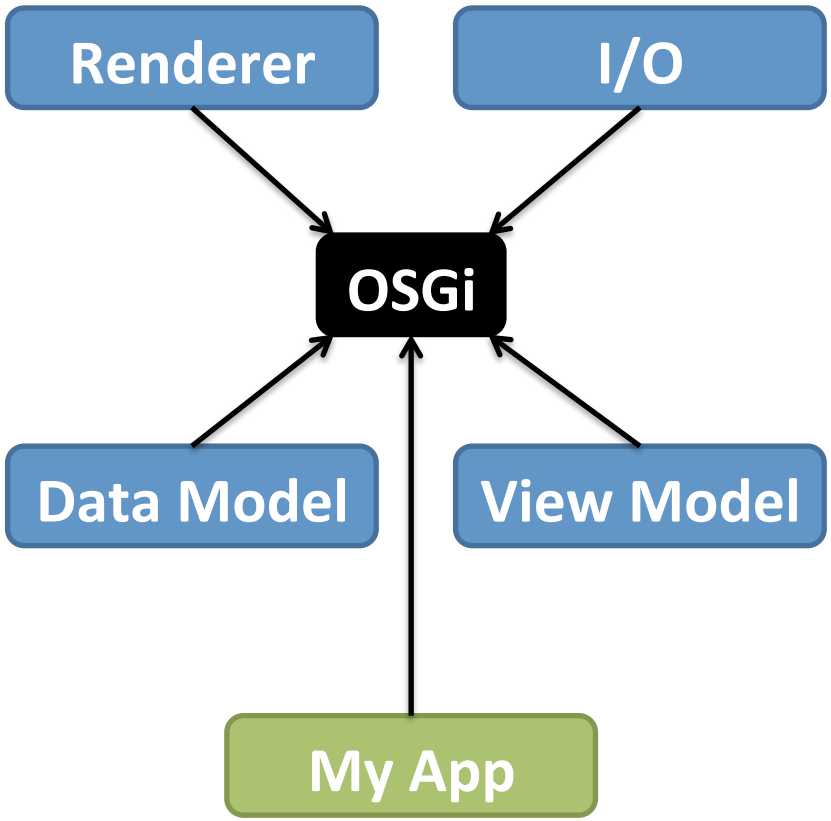
Anatomy of a Bundle

- JAR with extra metadata
- Imports
  - The Java packages used by the bundle
- Exports
  - Java packages in the bundle that other bundles are allowed to use (usually just API)
- Activator
  - Triggered when bundle is started/stopped



Cytoscape 3

- Service-oriented microkernel
- OSGi core
  - Dynamically loads/ unloads modules, a.k.a. bundles
- Each subsystem in Cy3 has separate OSGi bundle(s)
- Apps can also be packaged as bundles



Example: HelloWorld

- `pom.xml`
  - Maven project descriptor
  - Maven identifier
    - Group id
    - Artifact id
    - Version
  - OSGi identifier
    - Bundle-SymbolicName
  - Describes imports/ exports
- `Activator.java`
  - Bundle activator

Maven Project Layout

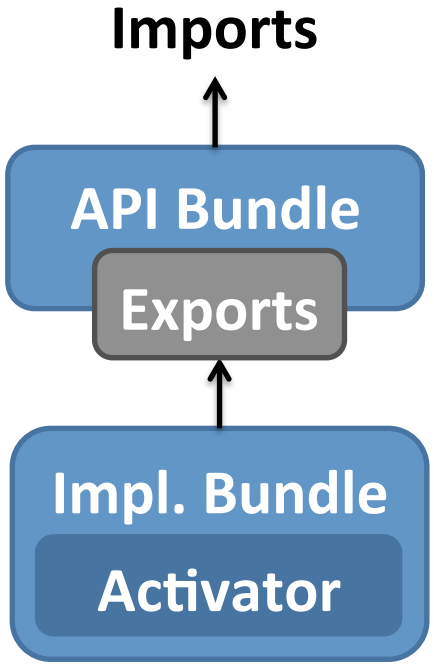
- `pom.xml`
  - Project descriptor
- `src/main/java`
  - Bundle code
- `src/test/java`
  - Test code
  - Not included in bundle JAR
- `src/main/resources`
  - Non-code files that should be included in bundle JAR

Core Bundles

- app
- application
- command-executor
- core-task
- custom-graphics
- datasource
- equations
- event
- group
- io
- layout
- model
- presentation
- property
- service
- session
- swing-util
- viewmodel
- vizmap
- vizmap-gui
- webservice
- work

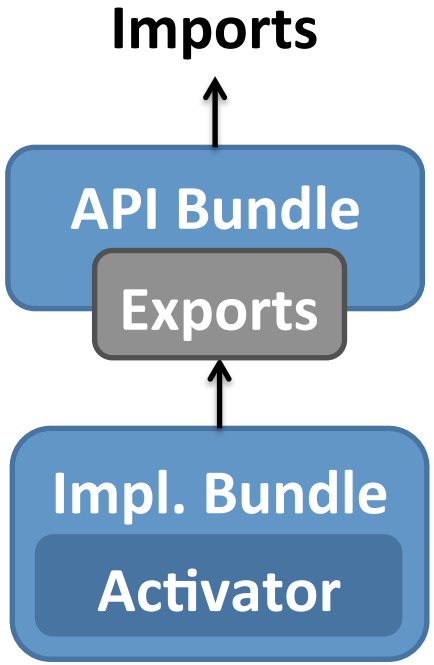
Core Bundles

- Usually come in sets:
  - API (optional)
    - No activator
  - Implementation
    - At least one per API bundle
    - No exports
- Separate API so we can keep implementation modular
  - Desktop application
  - Console application, for scripts



Core Bundles

- Nothing should import implementation bundles
- Unless it's for unit testing

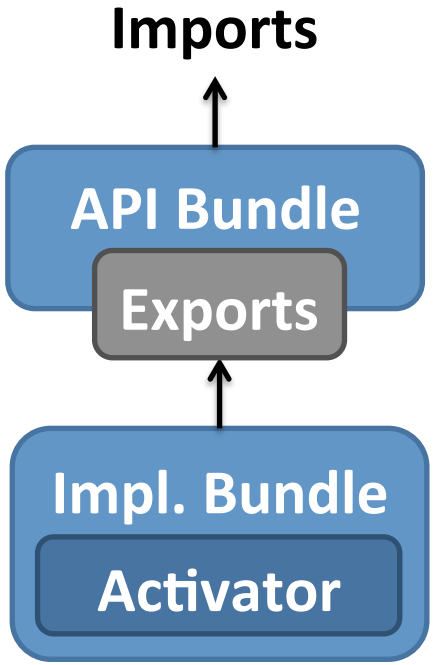


Core Bundles

- Task bundles
  - **work-api**
  - **work-swing-api**
  - work-impl
  - work-swing-impl
  - work-headless-impl
- VizMapper bundles
  - **vizmap-api**
  - **vizmap-gui-api**
  - vizmap-gui-core-impl
  - vizmap-gui-impl
  - vizmap-impl

OSGi Services

- Service
  - An instance of a Java interface
  - The glue behind API and implementation bundles
  - Usually registered by a BundleActivator
- In Cy3:
  - Defined by an API bundle
  - Registered by an implementation bundle



OSGi Services

- Interface
- Implementation
- Properties
  - Arbitrary key-value pairs

- `interface MyService { … }`
- `class MyServiceImpl implements MyService { … }`
- `(“title”, “My Service”) (“preferredMenu”, “Apps”)`

Cytoscape API

- Available as OSGi services
- Two main types:
  - API: Application Programming Interface
    - Just fetch and use:

```
MyService service = getService(context, MyService.class);
```

  - SPI: Service Provider Interface
    - Implement/extend and register:

```
registerService(context, new MyServiceImpl(),
                MyService.class, properties);
```

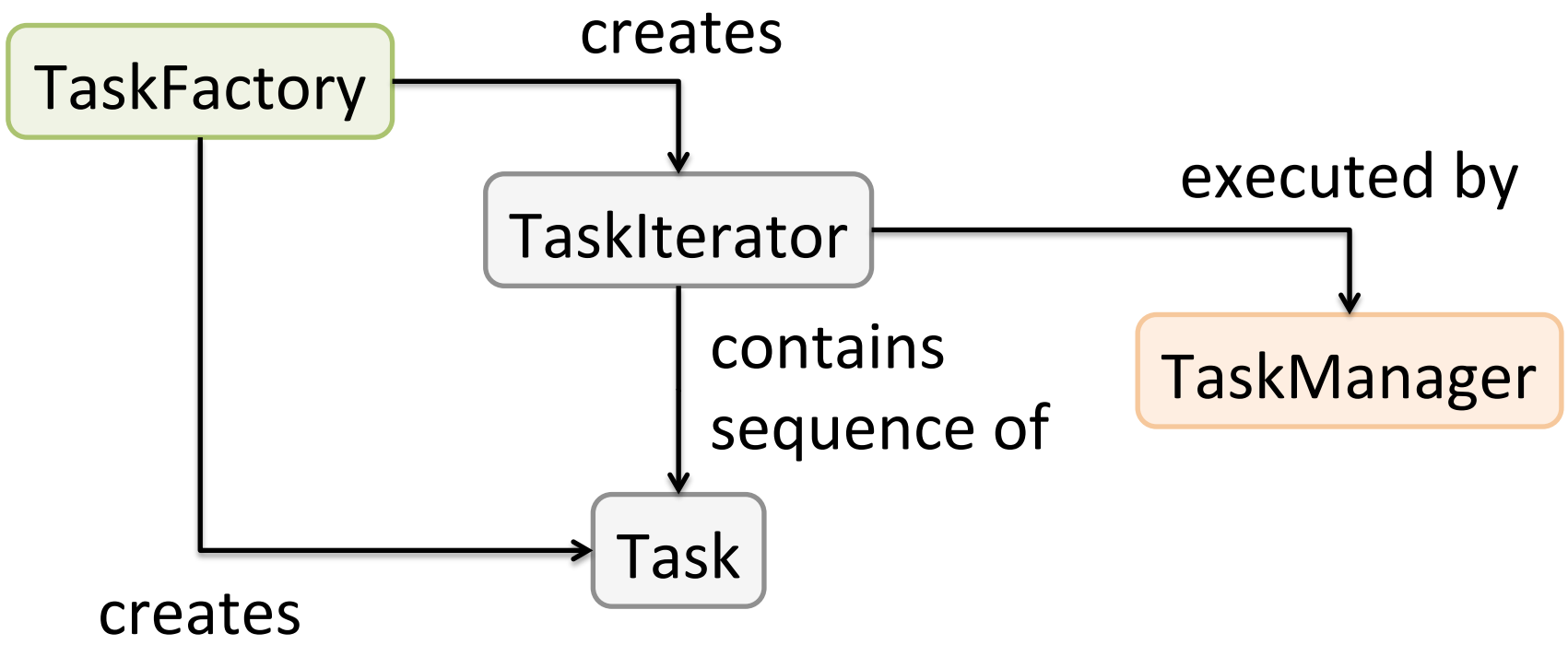
OSGi Services

- Most common types of services:
  - Factories
    - Create new instances
  - Managers
    - Track, provide access to, or operate on collections of objects
  - Utilities
    - Collections of utility functions

Cytoscape TaskFactories

- TaskFactories
  - Main unit of work in Cytoscape is a “Task”
  - Tasks are created by TaskFactories

Tasks


TaskFactory
creates
TaskIterator
executed by
TaskManager
contains
sequence of
Task
creates

Cytoscape TaskFactories

- TaskFactories
  - Main unit of work in Cytoscape is a "Task"
  - Tasks are created by TaskFactories
  - TaskFactories are OSGi services
  - Can be registered in your CyActivator:

```
TaskFactory factory =
        registerService(bc, myFactory, TaskFactory.class, properties);
```

where myFactory is the task factory you want to register

  - properties provide meta-data about the factory
    - Java Properties

Cytoscape TaskFactories

- Cytoscape TaskFactory Properties
  - Properties have special meaning in Cytoscape
  - Defined in org.cytoscape.work.ServiceProperties
  - Key properties
    - TITLE – If used as a menu, this is the menu title
    - PREFERRED_MENU – Where this will be added
    - ENABLE_FOR – When this menu is active
    - IN_TOOL_BAR – is it in the tool bar?
    - IN_MENU_BAR – is it in the top-level menus?
    - MENU_GRAVITY – The specific gravity of this item.

Cytoscape TaskFactories

- Example:

```
import org.cytoscape.work.AbstractTaskFactory;
import org.cytoscape.work.TaskIterator;

class MyTaskFactory extends AbstractTaskFactory {
        public MyTaskFactory() {
                super();
        }

        public TaskIterator createTaskIterator() {
                return null; // Fill in
        }

        public boolean isReady() { return true; }
}
```

Cytoscape TaskFactories

- Example (in CyActivator):

```
MyTaskFactory myFactory = new MyTaskFactory();
Properties props= new Properties();
// Note the "." notation for cascading menus
props.setProperty(PREFERRED_MENU, "Apps.cddApp");
props.setProperty(TITLE, "Load CDD Domains for Node");
// Not all task factories will be commands
props.setProperty(COMMAND, "loadCDDDomains4node");
props.setProperty(COMMAND_NAMESPACE, "cddApp");
props.setProperty(IN_MENU_BAR, "true");
// Usually means the second menu item
props.setProperty(MENU_GRAVITY, "2.0");
registerService(bc, loadCDDDomainNodeView,
                NodeViewTaskFactory.class, nodeViewProps);
```

Step 1: Project

- Add a new Cytoscape App menu
  - Menu title: Hello world!
  - For now, don't need a Task

Important Factories

- CyNetworkFactory
  - model-api
- CyTableFactory
  - model-api
- CyGroupFactory
  - group-api
- CyNetworkViewFactory
  - viewmodel-api
- VisualMappingFunctionFactory
  - vizmap-api
- VisualStyleFactory
  - vizmap-api

Important Managers

- CyApplicationManager
  - application-api
  - Lots of “state” information
- CyNetworkManager
  - model-api
- CyTableManager
  - model-api
- CyNetworkTableManager
  - model-api
  - Manages the association between tables and network objects
- CyNetworkViewManager
  - viewmodel-api
- CyGroupManager
  - group-api
- VisualMappingManager
  - vizmap-api

Special Task Factories

- core-task-api
  - NetworkViewTaskFactory
    - Network background context menu
  - NodeViewTaskFactory
    - Node context menu
  - EdgeViewTaskFactory
    - Edge context menu

Tasks

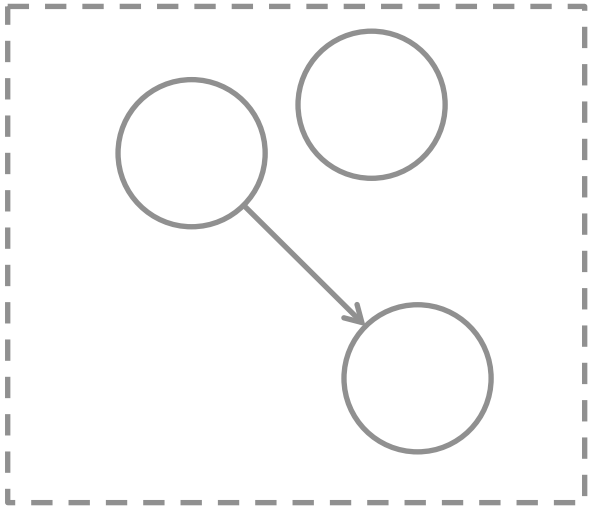
- Intended to run in multiple environments
  - Desktop application
  - Headless, via scripting
  - Programmatically, via an app
- Not appropriate to assume task will be run in a particular way
  - Only use Swing within tasks if necessary

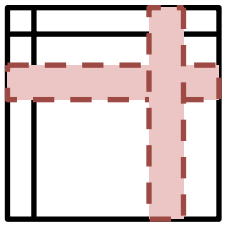
Step 2: Cytoscape Model

- Two basic concepts in data model:
  - CyNetwork
  - CyTable

Concepts: Data Model

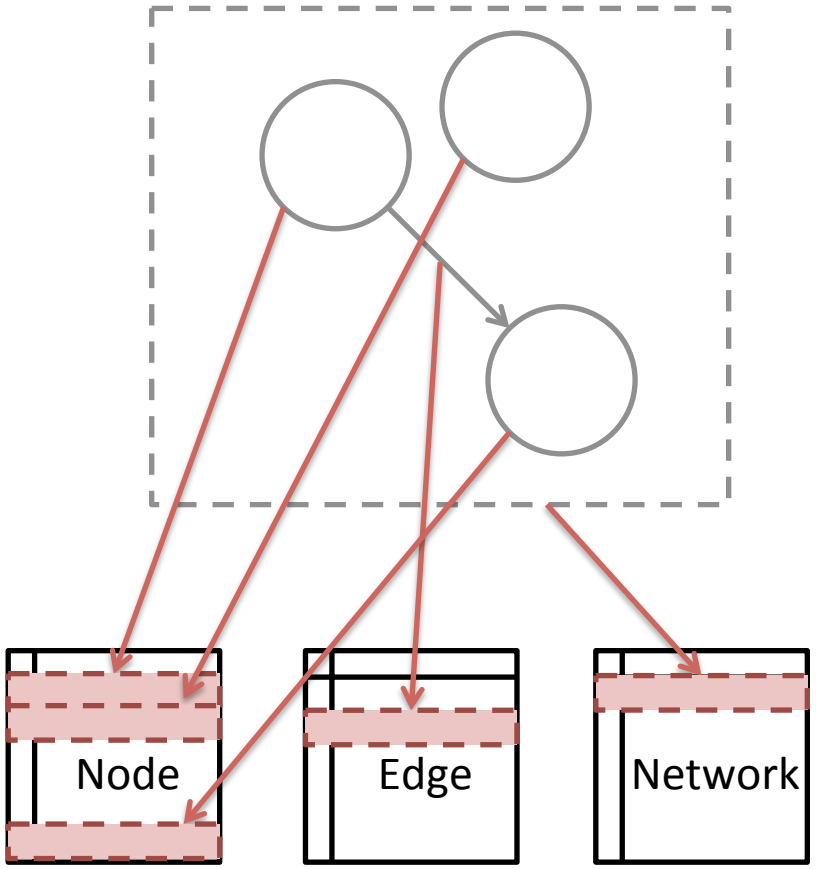
- CyNetwork
  - Multigraph
  - Directed or undirected
  - CyNode
  - CyEdge
- CyTable
  - CyColumn
    - Primary key
  - CyRow
- SUID
  - Session-unique identifier





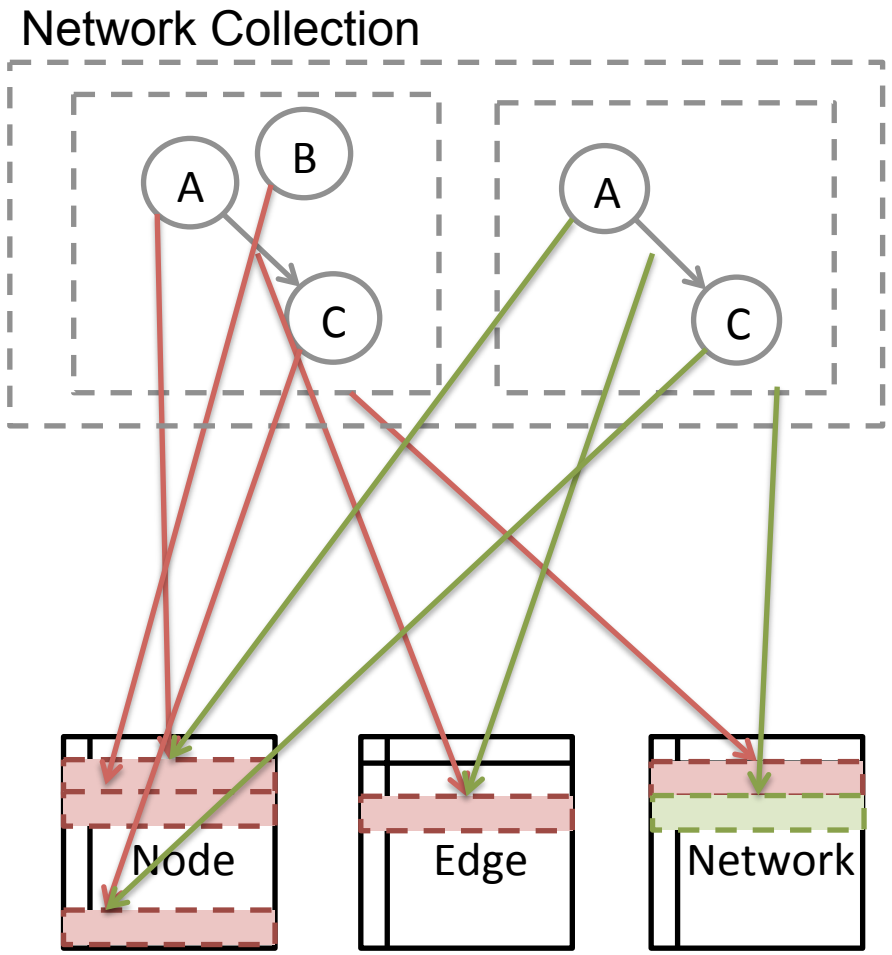
Concepts: Data Model

- CyNetwork
  - Multigraph
  - Directed or undirected
  - CyNode
  - CyEdge
- CyTable
  - CyColumn
    - Primary key
  - CyRow
- SUID
  - Session-unique identifier



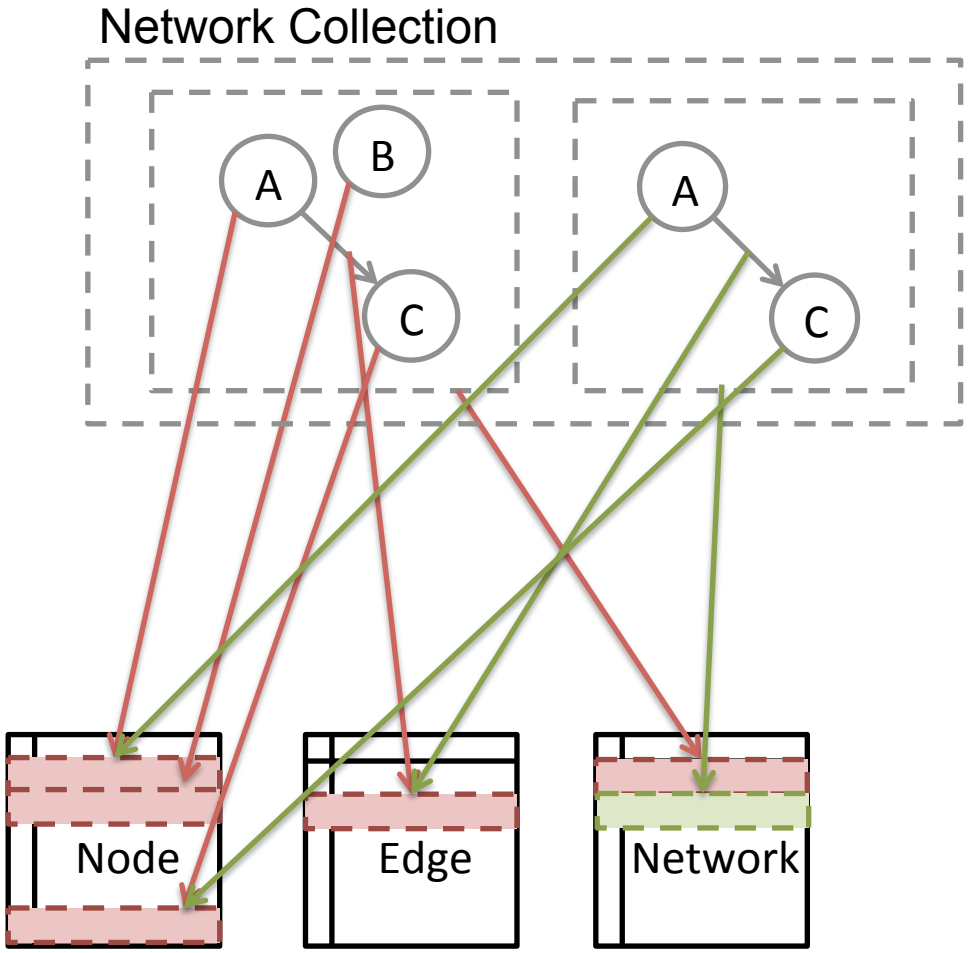
Concepts: Data Model

- CyNetwork
  - Can have multiple networks in a "collection"
    - Essentially a single-level hierarchy
  - CyRootNetwork
    - Top CyNetwork in the collection
    - Contains all nodes and edges
    - org.cytoscape.model.subnetwork
  - CySubNetwork
    - Projection of part of CyRootNetwork
    - Possibly multiple subnetworks
    - Nodes and edges have the same SUIDs as in CyRootNetwork
    - All CyNetworks not explicitly CyRootNetworks are CySubNetworks



Concepts: Data Model

- CyNetwork
  - Can have multiple networks in a "collection"
  - CyNode and CyEdge attributes can be shared
  - Network attributes are not shared
- CyTable
  - Actually have three "types" of tables:
    - Shared attributes
      - Shared within network collection
    - Local attributes
      - Not shared
    - "Façade" table
      - A view of the merged local and shared tables



Concepts: Data Model

- Creating a network
  - Use CyNetworkFactory to create the network and CyNetworkManager to add it
    - Common pattern
    - Need to get them in your CyActivator:

```
CyNetworkFactory networkFactory =
        getService(bc, CyNetworkFactory.class);
CyNetworkManager networkManager =
        getService(bc, CyNetworkManager.class);
```

    - Then pass them to your task factory in its constructor
  - Then just use them:

```
CyNetwork newNetwork = networkFactory.createNetwork();
networkManager.addNetwork(newNetwork);
```

  - Most operations (including adding nodes and edges) are on CyNetwork

Step 2: Project

- Add a task to the TaskFactory from Step 1
- Task should add a node to the network
  - I recommend that your Task extends AbstractTask
  - If you have time, add two nodes and one edge between them
- NOTE: You will have to create a view manually
- HINT: you will need to edit your pom.xml file to include the additional dependency

Step 3: Table Model

- CyTables
  - Standard table model:
    - Columns are fixed-type:
      - Boolean, String, Integer, Long, Double
      - List<Boolean>, List<String>, List<Integer>, List<Long>, List<Double>
    - Rows are singly indexed by a key
  - Columns can be “virtual”
    - Essentially functions as a link from one table into another

Tables and Networks

- Creating a CyNetwork creates:
  - Network tables
    - LOCAL_ATTRS and HIDDEN_ATTRS for each network
  - Node tables
    - LOCAL_ATTRS and HIDDEN_ATTRS each network
    - DEFAULT_ATTRS for each network
      - » All columns except key are virtual (Combines LOCAL_ATTRS and SHARED_ATTRS
    - SHARED_ATTRS for each collection
  - Edge tables
    - LOCAL_ATTRS, HIDDEN_ATTRS, and DEFAULT_ATTRS for each network
    - SHARED_ATTRS for each collection

Tables and Networks

Model Object	Table	Key	Notes
CyNetwork	LOCAL_ATTRS	CyNetwork.SUID	Standard local table
CyNetwork	HIDDEN_ATTRS	CyNetwork.SUID	Not shown to user
CyNode	LOCAL_ATTRS	CyNode.SUID	Local table not shared across networks in the same collection
CyNode	SHARED_ATTRS	CyNode.SUID	Shared table. One for each network collection.
CyNode	DEFAULT_ATTRS	CyNode.SUID	The façade table. Essentially all virtual columns pointing to LOCAL_ATTRS and SHARED_ATTRS tables
CyNode	HIDDEN_ATTRS	CyNode.SUID	Local attributes not shown to the user
CyEdge	LOCAL_ATTRS	CyEdge.SUID	Local table not shared across networks in the same collection
CyEdge	SHARED_ATTRS	CyEdge.SUID	Shared table. One for each network collection.
CyEdge	DEFAULT_ATTRS	CyEdge.SUID	The façade table. Essentially all virtual columns pointing to LOCAL_ATTRS and SHARED_ATTRS tables
CyEdge	HIDDEN_ATTRS	CyEdge.SUID	Local attributes not shown to the user

Table Model

- Standard columns
  - CyNetwork.NAME
    - Name of the node, edge, or network
  - CyNetwork.SELECTED
    - If TRUE, this object is selected
  - CyRootNetwork.SHARED_NAME
    - In the SHARED_ATTRS tables
    - Shared (root) name of the object

Table Model

- Can also create your own tables
  - Bound to network objects
    - Key is *always* SUID (Long)
    - Should be registered with CyNetworkTableManager
    - Can be easily pulled from CyNetwork:

```
CyNetwork.getTable(Class<? extends CyIdentifiable> type, String namespace);
```

    - Shortcut to get a row:

```
CyNetwork.getRow(CyIdentifiable entry, String namespace);
```

  - Unbound
    - Key is any valid type
    - Should be registered with CyTableManager

Accessing Tables

- Testing for columns
  - Must test for column existence before access

```
if (table.getColumn(String columnName) != null)
```

- Creating new columns
  - Columns must be typed
  - List columns must include the list type
- Getting rows
  - From table:

    getRow(Object key)
  - From a network:

    getRow(CyIdentifiable entry, String namespace)

Accessing Tables

- Getting data
  - All data access is through rows:

```
<T> T CyRow.get(String columnName, Class<? extends T> type)
Integer I = row.get("clusterNumber", Integer.class);
```

  - Where "type" is the column type. It is an error if the type is wrong
- Setting data

```
CyRow.set(String columnName, T value)
```

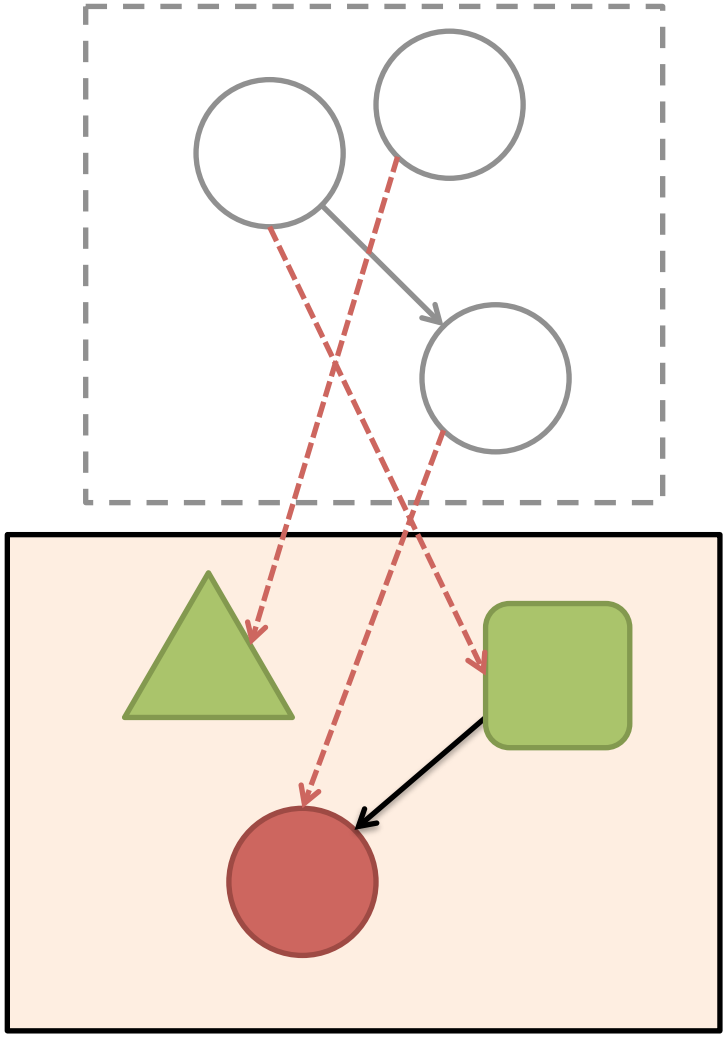
  - Where T is the column type
- Note: adding nodes and edges automatically adds the corresponding rows

Step 3: Project

- In the network you created before:
  - Change the name of the node or nodes
  - Create two new node columns:
    - Hello → List of Strings
    - World → Double
  - Add data to the new columns

Step 4: View Model

- CyNetworkView
  - View<CyNode>
  - View<CyEdge>
- VisualProperty
  - Examples:
    - NODE_X_LOCATION
    - EDGE_WIDTH
    - NETWORK_HEIGHT



View Model

- Creating a network view

  CyNetworkViewFactory.createNetworkView(CyNetwork network);

  - Need to get CyNetworkViewFactory in your CyActivator
  - Will create Views for all nodes and edges

- Getting a network view

  CyNetworkViewManager.getNetworkViews(CyNetwork network);

  - Need to get CyNetworkViewManager in your CyActivator
  - Note you get a collection of views back – i.e. there can be multiple views per network

- Getting node and edge views

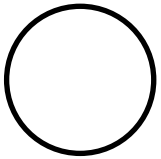
  View<CyEdge> edgeView = CyNetworkView.getNodeView(CyNode node);

  View<CyNode> nodeView = CyNetworkView.getEdgeView(CyEdge edge);

Visual Properties

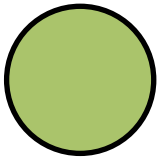
- VisualLexicon
  - VisualProperty hierarchy
  - Child properties inherit values from parents

- Node
  - NODE_PAINT
    - NODE_BORDER_PAINT
    - NODE_FILL_COLOR



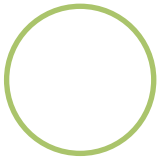
Visual Properties

- VisualLexicon
  - VisualProperty hierarchy
  - Child properties inherit values from parents
- Node
  - NODE_PAINT
    - NODE_BORDER_PAINT
    - **NODE_FILL_COLOR**



Visual Properties

- VisualLexicon
  - VisualProperty hierarchy
  - Child properties inherit values from parents
- Node
  - NODE_PAINT
    - **NODE_BORDER_PAINT**
    - NODE_FILL_COLOR



Visual Properties

- VisualLexicon
  - VisualProperty hierarchy
  - Child properties inherit values from parents
- Node
  - **NODE_PAINT**
    - NODE_BORDER_PAINT
    - NODE_FILL_COLOR

Setting a Visual Property

- Visual properties can be directly set in View interface:

```
view.setLockedValue(VisualProperty<? extends T> vp, V value)
```

…where View is a node view (View<CyNode>) or an edge view (View<CyEdge>)

- Usually get the VisualProperty from BasicVisualLexicon
- Types are important. Need to make sure V is an appropriate type for the VisualProperty
- Example:

```
nodeView.setLockedValue(BasicVisualLexicon.NODE_FILL_COLOR,
                        Color.BLUE);
```

Visual Styles

- VisualMappingFunction
  - Maps between a CyColumn value and a VisualProperty value
    - E.g. “name” column mapped to NODE_LABEL
- VisualStyle
  - Collection of VisualMappingFunctions

- Three mapping types:
  - Passthrough
    - Usually used for labels
  - Discrete
    - Categorical data
  - Continuous
    - Range-to-range mappings
    - Node color gradient
    - Node size

Building a Visual Mapping

- VisualMappingManager
  - Manages all visual styles
  - Get it in your CyActivator as a service
- Getting the visual style for a network:

```
VisualStyle style =
    vmm.getVisualStyle(CyNetworkView networkView);
```

..where vmm is the VisualMappingManager

- Create a VisualStyle (usually make a copy)
  - Use VisualStyleFactory.createVisualStyle(style);
  - Get it in your CyActivator as a service

Building a Visual Mapping

- Get the desired VisualMappingFunctionFactory in your CyActivator:

```
VisualMappingFunctionFactory vmfFactoryC =
                getService(bc,VisualMappingFunctionFactory.class,
                           "(mapping.type=continuous)");
VisualMappingFunctionFactory vmfFactoryD =
                getService(bc,VisualMappingFunctionFactory.class,
                           "(mapping.type=discrete)");
VisualMappingFunctionFactory vmfFactoryP =
                getService(bc,VisualMappingFunctionFactory.class,
                           "(mapping.type=passthrough)");
```

- Note this is a little different. We're using the filter argument to getService

Building a Visual Mapping

- Passthrough:

```
//Use pass-through mapping
String ctrAttrName1 = "SUID";
PassthroughMapping pMapping = (PassthroughMapping)
          vmfFactoryP.createVisualMappingFunction(ctrAttrName1,
                                                  String.class,
                                                  BasicVisualLexicon.NODE_LABEL);
vs.addVisualMappingFunction(pMapping);

// Add the new style to the VisualMappingManager
vmm.addVisualStyle(vs);

// Apply the visual style to the NetworkView
vs.apply(myNetworkView);
myNetworkView.updateView();
```

Building a Visual Mapping

- Continuous:

```
// Set node color map to attribute "Degree”
ContinuousMapping mapping = (ContinuousMapping)
     vmfFactoryC.createVisualMappingFunction("Degree",
                                             Integer.class,
                                             BasicVisualLexicon.NODE_FILL_COLOR);
// Define the points
Double val1 = 2d;
BoundaryRangeValues<Paint> brv1 =
        new BoundaryRangeValues<Paint>(Color.RED, Color.GREEN, Color.GREEN);
Double val2 = 12d;
BoundaryRangeValues<Paint> brv2 =
        new BoundaryRangeValues<Paint>(Color.YELLOW, Color.YELLOW, Color.BLACK);
// Set the points
mapping.addPoint(val1, brv1);
mapping.addPoint(val2, brv2);

// add the mapping to visual style
vs.addVisualMappingFunction(mapping);
```

Visual Property Precedence

- How VisualProperty is determined for a View:
  1. Locked value from View (a.k.a. bypass)
  2. Mapped value from VisualStyle
  3. Default value for VisualStyle
  4. Default value for VisualProperty

Step 4: Project

- Modify your application to create an initial view
- Change the shape of your two nodes
- Create a visual style that:
  - Uses the first value in the Hello column to label the node
    - Will need to use a VisualBypass, not a passthrough mapper
  - Uses the value in the World column to set the color using a continuous mapper

Step 5: User Interface

- Tasks should work in both
  - Headless (i.e. nongui) mode
    - How do tasks get executed?
  - GUI mode
    - Separate Swing UI?
- Commands: headless mode
- Tunables: argument handling

Commands

- Exports TaskFactories to:
  - Command line dialog
  - REST interface
- Simple
  - Add to your TaskFactory properties:
    - COMMAND_NAMESPACE
      - Generally the name of your app.
      - All of your commands will be grouped under this
    - COMMAND
      - The command itself
- Arguments?

NOTE: want to be careful about what TaskFactories you export (nongui only)

Tunables

- Tunables:
  - are Java Annotations
  - automatically generate GUI
  - automatically expose command arguments
- Example:

```
import org.cytoscape.work.Tunable;
import org.cytoscape.work.AbstractTask;
class MyClass extends AbstractTask {
   @Tunable (description="My integer value")
   public int value;
}
```

Tunables

- Basic Types
  - int, double, float, String, long, boolean
  - File, URL
- Classes for more complicated Tunables
  - ListSingleSelection, ListMultipleSelection
  - BoundedDouble, BoundedFloat, BoundedInteger, BoundedLong
- Example:

```
@Tunable (description="Choose from the list")
public ListSingleSelection color=
     new ListSingleSelection("red", "blue", "green");
```

Tunables

- Command-only Tunables
  - Used for selection of nodes, edges, and rows
  - CyNetwork
  - Utility Classes
    - EdgeList, NodeList, RowList

Tunables

- Commonly used tunable parameters
  - context: limit tunable to certain context (“gui”, “nongui”, “**both**”)
  - dependsOn: dependency between tunables
  - gravity: control order of panels
  - groups: group tunable panels together
  - listenForChange: list of tunables that will update this tunable
  - Tooltip

Tunables

- Getters and Setters approach
  - Can also explicitly use getters and setters
  - Allows better control over values

```
@Tunable (description="Test")
public int getTest() { return value; }
public void setTest(int v) { value = v; }
```

  - Very useful for initialization values and reacting to changes

Tunables

- Odds and Ends
  - Can have context classes with multiple tunables
    - ContainsTunables:

```
@ContainsTunables
public MyContext context;
```

    - Resulting UI will include context Tunables
  - Providing a title for the dialog
    - ProvidesTitle:

```
@ProvidesTitle
public String getTitle() {return “MyTitle”;}
```

Status Messages

- Two ways to inform users of status:
  - *org.cytoscape.work.TaskMonitor*
    - Passed as argument to run() method of Tasks
    - setTitle() and showMessage() provide status messages
      - Messages are also recorded in the Cytoscape Task History
    - setProgress() updates the progress bar
  - *org.cytoscape.application.CyUserLog*
    - General logging facility for user messages
      - Uses (or can use) Log4J
    - Messages are logged into the Cytoscape Task History

Step 5: Project

- Modify your Task:
  - Input the shape to make your nodes
  - Input the high and low colors for you range
- Export a command for your task

Step 6: Events

- Cytoscape philosophy:
  - Effect lower layers by method invocation
  - Effect upper layers by event handling
- Lower layers fire events to be handled by upper layers
- Look for ".events" packages:
  - org.cytoscape.model.events
  - org.cytoscape.view.model.events
  - org.cytoscape.application.events

Events

- Register your event listeners as services:

```
MyListener listener = new MyListener();
registerService(listener, NetworkAddedListener.class);
```

- Your class just needs to implement the appropriate handleEvent:

```
class MyListener implements NetworkAddedListener {
    public void handleEvent(NetworkAddedEvent ev) {
    // handle your data
    }
}
```

Events (Selection)

- Listening for selection
  - (we really didn't try to make this hard…
  - …but we succeeded)
- Programmatically selecting nodes or edges:
  - Set CyNetwork.SELECTED to True in the appropriate default table
- Listening for selection:
  - Listen for changes to rows (RowsSetListener)

RowsSetListener

- Probably want either
  - The networks you care about, or
  - CyNetworkTableManager

  …in your constructor
- Implement RowsSetListener
- RowsSetEvent:
  - source is the CyTable that contains the changed rows
  - getColumnRecords(String column) returns a collection of RowSetRecords
  - Finally, the each RowSetRecord has the row, column, and value

```
public class NetworkSelectionLinker implements RowsSetListener {
    // Define variables
public NetworkSelectionLinker(CyRootNetwork rootNetwork, CyEventHelper eventHelper) {
    this.rootNetwork = rootNetwork;
    this.eventHelper = eventHelper;
    this.viewManager = clusterManager.getService(CyNetworkViewManager.class);
  }

  public void handleEvent(RowsSetEvent e) {
    if (!e.containsColumn(CyNetwork.SELECTED) || ignoreSelection)
      return;
    CyNetworkView currentNetworkView = clusterManager.getNetworkView();
    ignoreSelection = true;
    Map<CyNetwork, Boolean> stateMap = new HashMap<CyNetwork, Boolean>();
    for (CySubNetwork subNetwork: rootNetwork.getSubNetworkList()) {
      if (e.getSource().equals(subNetwork.getTable(CyNode.class, CyNetwork.LOCAL_ATTRS))) {
        for (RowSetRecord record: e.getColumnRecords(CyNetwork.SELECTED)) {
          Long suid = record.getRow().get(CyIdentifiable.SUID, Long.class);
          Boolean value = (Boolean)record.getValue();
          for (CySubNetwork sub2: rootNetwork.getSubNetworkList()) {
            if (subNetwork.equals(sub2) || sub2.getDefaultNodeTable().getRow(suid) == null)
              continue;
            sub2.getDefaultNodeTable().getRow(suid).set(CyNetwork.SELECTED, value);
          }
        }
      }
      if (viewManager.viewExists(subNetwork)) {
        for (CyNetworkView view: viewManager.getNetworkViews(subNetwork)) {
          if (!view.equals(currentNetworkView)) { view.updateView(); }
        }
      }
    }
    eventHelper.flushPayloadEvents();
    ignoreSelection = false;
  }
}
```

Step 6: Project

- Add a selection listener to your app
  - When a node is selected, change the shape

Step 7: Commands

- Loosely-coupled way to access functionality
  - Core
  - Apps
- General idea:
  - Commands are exported to the Command Tool
    - Tools->Command Line Dialog
  - Other apps can execute those commands using the `org.cytoscape.command` package

Commands

Available namespaces:

- cdd
- chemviz
- cluster
- clusterviz
- command
- edge
- gpml
- group
- layout
- network
- node
- rinalyzer
- seqViz
- session
- setsApp
- structureViz
- table
- view
- vizmap
- wikipathways

help network

Available commands:

structureViz align *Perform sequence-driven structural superposition on a group of structures.*

structureViz annotateRIN *Annotate a residue interaction network (RIN) with the attributes of the corresponding residues in Chimera.*

structureViz close

structureViz createRIN *Create a residue interaction network (RIN) from the current model(s) in Chimera.*

structureViz exit *Close all open models and exit Chimera*

structureViz launch *Launch Chimera.*

structureViz list models *List currently open Chimera models*.

structureViz open *Open new structures in Chimera*

structureViz send *Send a command to Chimera.*

structureViz set *Change structureViz settings*

structureViz showDialog *Show the molecular navigator dialog*

structureViz syncColors *Synchronize colors between structure residues and network nodes.*

help structureViz open

structureViz open arguments*:*

edgeList=[*edgeColumn:value|edge name,...*]|all|selected|unselected: List of edges to open structures for

modbaseID=<*String*>: Modbase models to fetch

network=current|[*column:value|network name*]: Network for the selected nodes/edges

nodeList=[*nodeColumn:value|node name,...*]|all|selected|unselected: List of nodes to open structures for

pdbID=<*String*>: PDB ID to fetch

showDialog=true|false: Show the Molecular Structure Navigator dialog after opening the structure in Chimera

structureFile=<*File*>: Structure file

help network

Available commands:

network add *Add nodes and edges to a network (they must be in the current collection)*
network add edge *Add an edge between two nodes*
network add node *Add a new node to a network*
network clone *Make a copy of the current network*
network create *Create a new network*
network create attribute *Create a new attribute (column) in the network table*
network create empty *Create an empty network*
network delete *Delete nodes or edges from a network*
network deselect *Deselect nodes or edges in a network*
network destroy *Destroy (delete) a network*
network export *Export a network and its view to a file*
network get *Return a network*
network get attribute *Get the value for a network attribute*
network get properties *Get the visual property value for a network*
network hide *Hide nodes or edges in a network*
network import file *Import a network from a file*
network import url *Import a network from a URL*
network list *List all of the available networks*
network list attributes *List all of the attributes (columns) for networks*
network list properties *List all of the network visual properties*
network load file *Load a network file (e.g. XGMML)*
network load url *Load a network file (e.g. XGMML) from a url*
network rename *Rename a network*
network select *Select nodes or edges in a network*
network set attribute *Set a value in the network table*
network set current *Set the current network*
network set properties *Set network visual properties*
network show *Show hidden nodes and edges*

Executing Commands

1. See if the command is avilable
    1. Use `org.cytoscape.command.AvailableCommands`
2. Get a TaskManager
3. Populate Create a TaskIterator using the `CommandExecutorTaskFactory`
4. Use a `TaskObserver` to get results

```
// My Manager class
// From http://github.com/RBVI/StEMAPApp
public class StEMAPManager {
 final CyServiceRegistrar serviceRegistrar;
 final CyEventHelper eventHelper;
 CommandExecutorTaskFactory commandTaskFactory = null;
 SynchronousTaskManager taskManager = null;
 AvailableCommands availableCommands = null;

 public StEMAPManager(final CyServiceRegistrar cyRegistrar) {
    this.serviceRegistrar = cyRegistrar;
}

 public <S> S getService(Class<S> serviceClass) {
    return serviceRegistrar.getService(serviceClass);
 }

 public void executeCommand(String namespace, String command,
                            Map<String, Object> args, TaskObserver observer) {
    if (commandTaskFactory == null)
      commandTaskFactory = getService(CommandExecutorTaskFactory.class);
    if (availableCommands == null)
      availableCommands= getService(AvailableCommands.class);
    if (taskManager == null)
      taskManager = getService(SynchronousTaskManager.class);
    if (availableCommands.getNamespace(namespace) == null ||
        !availableCommands.getCommands(namespace).contains(command))
      throw new RuntimeException("Can't find command "+namespace+" "+command);
    TaskIterator ti = commandTaskFactory.createTaskIterator(namespace, command, args, observer);
    taskManager.execute(ti);
  }
```

```
// Load a PDB file into UCSF Chimer
// From http://github.com/RBVI/StEMAPApp
  public void loadPDB(String pdbPath, String extraCommands) {
    Map<String, Object> args = new HashMap<>();
    if (pdbPath != null)
      args.put("structureFile", pdbPath);
    else
      args.put("pdbID", getPDB());

    args.put("showDialog", "true");
    // Assumes that calling class implements TaskObserver
    // StructureViz will give us a text string containing the
    // name an number of the opened model
    executeCommand("structureViz", "open", args, this);

    try {
      // Wait for things to process
      Thread.sleep(500);
    } catch (Exception e) {}

    if (extraCommands != null) {
      args = new HashMap<>();
      args.put("command", extraCommands);
      executeCommand("structureViz", "send", args, null);
    }
  }
}
```

Step 7: Project

- Write a new app
  - Execute the command you exported in Step 5
  - Will need to set the values for the Tunables…

Sample Code

- Sample apps
  - https://github.com/cytoscape/cytoscape-samples
  - Can use as template for your own apps
- Real app example:
  - SIREN: SIgning of REgulatory Networks
  - http://baderlab.org/PegahKhosravi/SIREN
  - https://github.com/BaderLab/SirenApp
- Some apps on app store link to their source code (e.g. DynNetwork)

Sample Code

- UCSF RBVI repository
  - https://github.com/RBVI/

API Tour

- http://chianti.ucsd.edu/cytoscape-3.2.1/API/
- app-api
  - Simple app API (to help Cy2 plugin developers; not meant for bundle apps)
- core-task-api
  - Commonly used high-level tasks
  - e.g. loading networks/styles/tables, applying layouts

API Tour

- model-api
  - Network, table model
- event-api
  - Event model
- work-api
  - Tasks, TaskFactory
- layout-api
  - Defining layouts

API Tour

- presentation-api
  - Visual property definitions
- viewmodel-api
  - Setting visual properties
- vizmap-api
  - Visual mapping
- swing-util
  - GUI utilities; e.g. file load/save dialog; color chooser dialog

API Tour

- equations-api
  - Defining CyTable equations (like Excel functions)
- group-api
  - Working with CyGroups (a.k.a metanodes)
- io-api
  - Defining importers/exporters; reading streams
- (swing-)application-api
  - Accessing system-level state and events (e.g. UI panels, toolbar, menus, main JFrame)

API Tour

- property-api
  - Access/define system properties; Access session-level properties
- service-api
  - AbstractCyActivator; service (un)registration; service listener registration
- session-api
  - Access current session file name; take snapshot of current session

API Tour

- command-executor-api
  - Support for executing commands from tasks
- group-api
  - Support for CyGroups, including collapse/expand, attribute aggregation, and visualization options

Best Practices

- Bundles should minimize what they export
  - Don't make something API unless someone asks for it and you're ready to commit to it long term
- Bundle activators should do as little work as possible
  - Ideally, just register services
  - Do expensive initialization as lazily as possible
    - E.g. during menu activation

Miscellaneous

- Application state information
- GUI or not
- Dealing with lots and lots of service requests

Getting help

- cytoscape-helpdesk@googlegroups.com
- cytoscape-discuss@googlegroups.com
- scooter@cgl.ucsf.edu

Slides and Solutions

- Slides available at
  - http://www.cgl.ucsf.edu/home/scooter/Cytoscape3DevTut/ISMBslides.pdf
- My solutions available at:
  - http://www.cgl.ucsf.edu/home/scooter/Cytoscape3DevTut/solutions.zip

Conclusions

- You are now ready to write non-trivial Cytoscape apps
- This tutorial is part of the Cytoscape development ladder:
  http://wiki.cytoscape.org/Cytoscape_3/AppDeveloper/Cytoscape_App_Ladder
- You are now ½ of the way through the ladder!
- Many APIs not covered
  - Open source community: ~~steal~~ borrow from others
    - Always acknowledge the source

Questions?