Systrip developer manual

COLLABORATORS			
	TITLE : Systrip developer manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 30, 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Tables

Chapter 1

Introduction

With the development of high throughput methods in biology, efficient visualization tools are more and more required. In particular it helps in understanding biological shifts induced on an organism by environmental stresses. In that case the data generated covers a wide range of biological processes which can be gathered into a single network. This network generally contains hundreds of elements and, prior to any in depth analysis, biologists are used to reduce the scope of their studies by selecting a few relevant sub-networks. The visual analytic process thus follows a top down approach consisting in filtering out steady part of the network. To help users in this visual mining we propose a flexible environment called Systrip.

Systrip is a perspective dedicated to biological network analysis and takes full advantage of the Tulip library and plug-in system . More than simply using Tulip library, Systrip adds a network manipulation layer on top of it. Such architecture aims at encapsulating both of graph manipulation and user interaction libraries. Relying on a graph structure gives to Systrip a great flexibility allowing to support both relational and multi-dimensional data. It also simplifies future integration of different biological networks (e.g. gene regulatory network or protein-protein interaction network).

This document describes technical information for Systrip, as it is based on the Tulip library you can find more detailed information on Tulip documentation [1].

[1] See Tulip documentation web page

Chapter 2

Installation

2.1 Requirements

To work properly Systrip need some dependancies referenced in the next table.

Library	**Required version**
CMake	>= 2.6
Tulip	3.4.1
Qt	>= 4.6.x or Qt 4.7 with QtAssistantClient (not installed by default in Qt 4.7)
OpenBabel	>= 2.2.3
FFMPEG	0.5
GSoap	>= 2.7
libSBML	>= 4.3
MYSQL	-
LibXML2	-
zlib	-
freetype	-
glew	-
libjpeg	-
libpng	-
DoxyGen (for documentation only)	-
dblatex (for documentation only)	-

Table 2.1: Requirements table

Systrip is not a program itself, it's just an add-on for Tulip system, so installing Systrip on your system consists in fact to copy files into your Tulip installation directory. After installing Systrip run Tulip and you can use Systrip from the Tulip interface.

Plug-ins name	**Plug-ins type**	**Required version**
Document View	View	1.0
Histogram view	View	1.0
Parallel Coordinates view	View	1.0
Table view	View	1.0
Scatter Plot 2D view	View	1.0
FM^3 (OGDF)	Layout	1.0

Table 2.2: Tulip required plug-ins

To use Systrip plug-ins you had to have certain plug-ins not installed by default. To get these plugins use the Tulip plug-ins manager see the Tulip user manual chapter 4, section 3 for more information.

2.2 Compilation

2.2.1 Systrip compilation

The Systrip build process uses `CMake` to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` for each directory of the package. It may also create one or more `.h` files containing system-dependent definitions. To get CMake or for more information, see the CMake website.

You can add some option to the default build process with CMake.

Option	Type	Comments
COMPILE_TESTS	boolean	compile unitary tests. To run unitary tests just launch the `buildDirectory/controller/MetabolicNetworkExplorerController/Tests/Test` program. You need to have Systrip installed in Tulip before launching the tests. You may have to configure the TLP_DIR environnment variable to tell where the Tulip plugins are located, see Tulip documentation for more information on this variable.
COMPILE_DOC	boolean	Compile documentation. Documentation consist in a user manual, a developer manual and api documentation. Documentation can be found in the `buildDirectory/controller/docs` subdirectories.
TULIP_DIR	Path	Tulip installation directory.
OPENBABEL2_DIR	Path	OpenBabel installation directory.
GSOAP_PATH	Path	Gsoap installation directory.
LIBSBML_INCLUDE	Path	Libsbml includes directory.
LIBSBML_LIBRARY	File	Libsbml library.
FFMPEG_DIR	Path	FFMPEG installation directory.
MYSQL_DIR	Path	MYSQL installation directory.
CMAKE_BUILD_TYPE	None,Debug, Release,RelWithDebInfo,MinSizeRel	Control the build type

Table 2.3: Common CMake options for Systrip

THE SIMPLEST WAY TO COMPILE THIS PACKAGE IS:

1. Create a directory where the package will be build.
2. Configure the build process with cmake : `cmake buildDirectory sourceDirectory cmakeOptions`. Check the build configuration log to see if there is some errors.
3. Go to the build directory if you're not in.
4. Type `make` to compile the package.

5. Type `make install` to install Systrip inside the Tulip directory. On windows system files can be installed in the wrong directory and you had to move them manually. To check it go into the Tulip installation directory in the `lib` directory check if there is a bin directory, if it exists copy all its files into the `tlp` directory.
6. You can remove the program binaries and object files from the source code directory by typing `make clean`.
7. You can now launch Tulip and enjoy Systrip by launching the `tulip` program in the `bin` subdirectory of the installation directory. See user manual for more information on Systrip usage.

There is an issue when linking with the dynamic version of OpenBabel : when closing Systrip an error can occur during the destruction of OpenBabel plug-ins. This error don't affect the program usability. To fix it link with the static version of OpenBabel see "OpenBabel compilation" for more information.

2.2.2 LibSBML compilation

LibSBML uses the autotools to configure it's build. You don't need to set special build options to use it with Systrip.

THE SIMPLEST WAY TO COMPILE IT IS:

1. Go into the source directory of LIBSBML.
2. Configure the build process with the configure script : `./configure` . Check the build configuration log to see if there is some errors.
3. Type `make` to compile the package.
4. Type `make install` to install it.
5. You can remove the program binaries and object files from the source code directory by typing `make clean`.

If you need more help with LibSBML compilation see the it's website : http://sbml.org/Software/libSBML

2.2.3 OpenBabel compilation

OpenBabel uses Cmake as build system. To avoid the error when closing Systrip just set the BUILD_SHARED variable to OFF. If you're compiling on 64 bit system you will need to add `-fPIC` to `CMAKE_CXX_FLAGS`.

THE SIMPLEST WAY TO COMPILE THIS PACKAGE IS:

1. Create a directory where the package will be build.
2. Configure the build process with cmake :
   `cmake buildDirectory sourceDirectory cmakeOptions -DBUILD_SHARED=OFF.`
   Check the build configuration log to see if there is some errors.
3. Go to the build directory if you're not in.
4. Type `make` to compile the package.
5. Type `make install` to install OpenBabel.
6. You can remove the program binaries and object files from the source code directory by typing `make clean`.

Checks the OpenBabel site if you need more help for building it : http://openbabel.org/wiki/Main_Page

2.2.4 FFMPEG compilation

FFMPEG uses the autotools to configure it's build. There is some special options to set to use FFMPEG with Systrip : --enable-gpl --enable-memalign-hack --enable-swscale.

THE SIMPLEST WAY TO COMPILE IT IS:

1. Go into the source directory of FFMPEG.
2. Configure the build process with the configure script :
   `./configure --enable-gpl --enable-memalign-hack --enable-swscale`.
   Check the build configuration log to see if there is some errors.
3. Type `make` to compile the package.
4. Type `make install` to install it.
5. You can remove the program binaries and object files from the source code directory by typing `make clean`.

If you need more help with FFMPEG compilation see the it's website : http://ffmpeg.org/.

2.2.5 GSOAP compilation

You don't need to compile GSOAP to use it with Systrip. Just set the GSOAP_PATH to the source path of GSOAP.

Chapter 3

Plug-ins development

Systrip is based on Tulip plug-ins system. It enables to directly add new functionalities into the Systrip kernel.

One must keeps in mind that a plug-in have access to all the parts of Tulip. Thus, one must write plug-ins very carefully to prevent memory leak and also errors. A bug in plug-in can result in a "core dump" in the software that uses it. To enable the use of plug-ins, a program must call the initialization functions of the plug-ins. This function loads dynamically all the plug-ins and register them into a factory that will enable to directly access to it.

The PropertyAlgorithm class, is the class from which inherits different types of algorithms such as the BooleanAlgorithm class or the LayoutAlgorithm class (see picture below). This class is important in the way that every specific algorithm that you will develop will have to inherit from one of those classes. For example, if you write a plug-in to update the graph layout, your new class will have to inherit from the LayoutAlgorithm class which inherits from this PropertyAlgorithm class. [../doxygen/tulip-lib/classtlp_1_1PropertyAlgorithm__inherit__graph.png not found] Each one of the 8 classes presented above has a public member, *TypeName*`Property*` *typeName*`Result`, which is the data member that which have to be updated by your plug-in. After a successful run tulip will automatically copy this data member into the corresponding property of the graph. Following is a table showing the data member, graph property and 'Algorithms' GUI sub-menu corresponding to each subclass of Algorithm :

Class name	Data member	Graph property replaced
BooleanAlgorithm	booleanResult	viewSelection
ColorAlgorithm	colorResult	viewColor
DoubleAlgorithm	doubleResult	viewMetric
IntegerAlgorithm	integerResult	viewInt
LayoutAlgorithm	layoutResult	viewLayout
SizeAlgorithm	sizeResult	viewSize
StringAlgorithm	stringResult	viewLabel

3.0.6 Overview of the class

A quick overview of the functions and data members of the class PropertyAlgorithm is needed in order to have a generic understanding of its 8 derived classes.

3.0.6.1 Public members

Following is a list of all public members :

- `PropertyAlgorithm (const PropertyContext& context) :`
  The constructor is the right place to declare the parameters needed by the algorithm.
  ```
  addParameter<DoubleProperty>("metric", paramHelp[0], 0, false);
  ```
  And to declare the algorithm dependencies.
  ```
  addDependency<Algorithm>("Quotient Clustering", "1.0");
  ```

- `~PropertyAlgorithm ()` : Destructor of the class.
- `bool run ()` :

  This is the main method :

  - It will be called out if the pre-condition method (bool check (..)) returned true.

  - It is the starting point of your algorithm.

  The returned value must be true if your algorithm succeeded.
- `bool check (std::string& errMsg)` :

  This method can be used to check what you need about topological properties of the graph, metric properties on graph elements or anything else.

3.0.6.2 Protected members

Following is a list of all protected members :

- `Graph* graph` :

  This graph is the one given in parameters, the one on which the algorithm will be applied.
- `PluginProgress* pluginProgress` :

  This instance of the class PluginProgress can be used to have an interaction between the user and our algorithm. See the next section for more details.
- `DataSet* dataSet` :

  This member contains all the parameters needed to run the algorithm. The class DataSet is a container which allows insertion of values of different types. The inserted data must have a copy-constructor well done. See the section called DataSet for more details.

The methods of the *TypeName*Algorithm class, will be redefined in your plug-in as shown in Section 3.0.9.

3.0.7 Parameters :

Your algorithm may need some parameters, for example a boolean or a property name, that must be filled in by the user just before being launched. In this section, we will look at the methods and techniques to do so.

3.0.7.1 Adding parameters to an algorithm

The class PropertyAlgorithm inherits from a class called WithParameters that has a member function named `template<typename Type> void addParameter (const char *name, const char *inHelp=0, const char *inDefValue=0, bool isMandatory=true)` which is capable of adding a parameter. This method has to be called in the constructor of your class.

Following is a description of its parameters :

- name : Name of the new parameter.
- inHelp : This parameter can be used to add a documentation to the parameter (See example below).
- inDefValue : Default value.
- isMandatory : If false, the user must give a value.

On the following example, we declare a character buffer that will contain the documentation of our parameters.

```
namespace {
  const char * paramHelp[] = {
    // property
    HTML_HELP_OPEN() \
    HTML_HELP_DEF( "type", "DoubleProperty" ) \
    HTML_HELP_BODY() \
    "This metric is used to affect scalar values to graph items." \
    "The meaning of theses values depends of the choosen color model." \
    HTML_HELP_CLOSE(),
    // colormodel
    HTML_HELP_OPEN() \
    HTML_HELP_DEF( "type", "int" ) \
    HTML_HELP_DEF( "values", "[0,1]" ) \
    HTML_HELP_DEF( "default", "0" ) \
    HTML_HELP_BODY() \
    "This value defines the type of color interpolation. Following values are valid :" \
    "<ul><li>0: HSV interpolation ;</li><li>1: RGB interpolation</li></ul>" \
    HTML_HELP_CLOSE(),
    // color1
    HTML_HELP_OPEN() \
    HTML_HELP_DEF( "type", "Color" ) \
    HTML_HELP_DEF( "values", "[0,255]^4" ) \
    HTML_HELP_DEF( "default", "red" ) \
    HTML_HELP_BODY() \
    "This is the start color used in the interpolation process." \
    HTML_HELP_CLOSE(),
    // color2
    HTML_HELP_OPEN() \
    HTML_HELP_DEF( "type", "Color" ) \
    HTML_HELP_DEF( "values", "[0,255]^4" ) \
    HTML_HELP_DEF( "default", "green" ) \
    HTML_HELP_BODY() \
    "This is the end color used in the interpolation process." \
    HTML_HELP_CLOSE(),
    // Mapping type
    HTML_HELP_OPEN() \
    HTML_HELP_DEF( "type", "Boolean" ) \
    HTML_HELP_DEF( "values", "true / false" ) \
    HTML_HELP_DEF( "default", "true" ) \
    HTML_HELP_BODY() \
    "This value defines the type of mapping. Following values are valid :" \
    "<ul><li>true : linear mapping</li><li>false: uniform quantification</li></ul>" \
    HTML_HELP_CLOSE(),
  };
}
```

Then, we can add the parameters in the constructor by writing the following lines:

```
    addParameter<DoubleProperty>("property",paramHelp[0],"viewMetric");
    addParameter<int>("colormodel",paramHelp[1],"1");
    addParameter<bool>("type",paramHelp[4],"true");
    addParameter<Color>("color1",paramHelp[2],"(255,255,0,128)");
    addParameter<Color>("color2",paramHelp[3],"(0,0,255,228)");
```

The picture below is the result of the sample of code above. [images/plugin_doc_add_parameter_1.png not found]

3.0.7.2 Accessing a parameter

The class PropertyAlgorithm has a protected member called *dataSet* that contains all the parameters value.The DataSet class implements a container which allows insertion of values of different types and implements the following methods :

- `template<typename T> bool get (const std::string& name, T value) const` :
  Returns a copy of the value of the variable with name name. If the variable name doesn't exist return false else true.
- `template<typename T> bool getAndFree (const std::string& name, T value)` :
  Returns a copy of the value of the variable with name name. If the variable name doesn't exist return false else true. The data is removed after the call.
- `template<typename T> void set (const std::string& name, const T value)` :
  Set the value of the variable name.
- `bool exist (const std::string& name) const` :
  Returns true if name exists else false.
- `Iterator<std::pair<std::string, DataType> >* getValues () const` :
  Returns an iterator on all values

Has you could have guess, the one important to access a parameter is get() which allows to access to a specific parameter. Following is an example of its use :

```
DoubleProperty* metricS;
int colorModel;
Color color1;
Color color2;
bool mappingType = true;

if ( dataSet!=0 ) {
  dataSet->get("property", metricS);
  dataSet->get("colormodel", colorModel);
  dataSet->get("color1", color1);
  dataSet->get("color2", color2);
  dataSet->get("type", mappingType);
}
```

3.0.8 The PluginProgress class.

The class PluginProgress can be used to interact with the user. Following is a list of its members

3.0.8.1 Public members

Following is a list of all Public members :

- `ProgressState progress (int step, int max_step)` :
  This method can be used to know the global progress of or algorithm, the number of steps accomplished.
- `void showPreview (bool)` :
  Enables to specify if the preview check box has to be visible or not.
- `bool isPreviewMode ()` :
  Enables to know if the user has checked the preview box.

- `ProgressState state () const :`
  Indicates the state of the 'Cancel', 'Stop' buttons of the dialog
- `void setError (std::string error) :`
  Shows an error message to the user
- `void setComment (std::string msg) :`
  Shows a comment message to the user

3.0.8.2 PluginProgress example

In following small example, we will iterate over all nodes and notify the user of the progression.

```
unsigned int i=0;
unsigned int nbNodes = graph->numberOfNodes ();

const unsigned int STEP = 10;

node n;
forEach(n, graph->getInEdges(n))
{
  ...
  ... // Do what you want
  ...
  if(i%STEP==0)
  {
     pluginProgress->progress(i, nbNodes); //Says to the user that the algorithm has ↩
         progressed.

     //exit if the user has pressed on Cancel or Stop
     if(pluginProgress->state() != TLP_CONTINUE)
     {
        returnForEach pluginProgress->state()!=TLP_CANCEL;
     }
  }
  i++;

}
```

Before exiting, we check if the user pressed stop or cancel. If he pressed "cancel", the graph will not be modified. If he pressed "stop", all values computed till now will be saved to the graph.

3.0.9 Example of a plug-in skeleton

Following is an example of a dummy color algorithm (you can find more example in the tulip tarball):

```
#include <tulip/TulipPlugin.h>
#include <string>

using namespace std;
using namespace tlp;

/** Algorithm documentation */
// MyColorAlgorithm is just an example
/*@{*/
```

```
class MyColorAlgorithm:public ColorAlgorithm {
public:

  // The constructor below has to be defined,
  // it is the right place to declare the parameters
  // needed by the algorithm,
  // addParameter<DoubleProperty>("metric", paramHelp[0], 0, false);
  // and declare the algorithm dependencies too.
  // addDependency<Algorithm>("Quotient Clustering", "1.0");
  MyColorAlgorithm(const PropertyContext& context):ColorAlgorithm(context) {
  }

  // Define the destructor only if needed
  // ~MyColorAlgorithm() {
  // }

  // Define the check method only if needed.
  // It can be used to check topological properties of the graph,
  // metric properties on graph elements or anything else you need.
  // bool check(string& errorMsg) {
  //   errorMsg="";
  //   return true;
  // }

  // The run method is the main method :
  //     - It will be called out if the pre-condition method (bool check (..)) returned  ↩
      true.
  //     - It is the starting point of your algorithm.
  // The returned value must be true if your algorithm succeeded.
  bool run() {
    return true;
  }
};
/*@}*/

// This line is very important because it's the only way to register your algorithm in  ↩
    tulip.
// It automatically builds the plug-in object that will embed the algorithm.
COLORPLUGIN(MyColorAlgorithm, "My Color Algorithm", "Authors", "07/07/07", "Comments",  ↩
    "1.0")
// If you want to present your algorithm in a dedicated sub-menu of the Tulip GUI,
// use the declaration below where the last parameter specified the name of sub-menu.
// COLORPLUGINOFGROUP(MyColorAlgorithm, "My Color Algorithm", "Authors", "07/07/07", " ↩
    Comments", "1.0", "My algorithms");
```

3.0.10 Metabolic network specific properties

If you develop plug-ins for Systrip you can access to certain properties loaded from the SBML file, these properties contains useful information on the network. The next table describes the list of these properties with their types and description.

Property name	Property type	Description
boundaryCondition	BooleanProperty	Only for compounds nodes.Can have "true" or "false" values. Define if the compounds is a boundary species i.e if it's quantity is not determined by the set of reactions even when that species occurs as a product or reactant. If true the species is on the boundary of the reaction system, and its quantity is not determined by the reactions.
charge	IntegerProperty	The charge property takes an integer indicating the charge on the species (in terms of electrons, not the SI unit coulombs).
constant	BooleanProperty	Indicates whether the species' quantity can be changed at all, regardless of whether by reactions, rules, or constructs other than InitialAssignment.
compartment	StringProperty	Only for compounds nodes. The compartment represents a bounded space in which species are located.
ecNumber	StringProperty	Only for reaction nodes. Correspond to the Enzyme Commission number (EC number) for the reaction. The EC number is a numerical classification scheme for enzymes, based on the chemical reactions they catalyze.
elmentType	IntegerProperty	Indicates the node type. If the value is 0 the element type is undefined, 1 is for species and 2 for reactions.
fastReaction	BooleanProperty	See the fast attribute in the SBML specification.
geneAssociation	StringProperty	
hasOnlySubstanceUnits	BooleanProperty	See the hasOnlySubstanceUnits attribute in the SBML specification.
id	StringProperty	The id of the elements from SBML file. Each nodes have unique id.
initialAmount	DoubleProperty	Values are defined only for compounds nodes. It represents species' initial quantity.
initialConcentration	DoubleProperty	See initialConcentration attribute in SBML specification.
listOfProducts	IntegerVectorProperty	Only for reaction nodes. For each reaction node this property contain the list of it's products. Each integer in the list correspond to a node id.
listOfReactants	IntegerVectorProperty	Only for reaction nodes. For each reaction node this property contain the list of it's reactants. Each integer in the list correspond to a node id.
name	StringProperty	The name of the elements.
proteinAssociation	StringProperty	
reaction	BooleanProperty	For each nodes. Defines the type of the element, if set to true the node is a reaction else it's a compounds.

Property name	Property type	Description
reversible	BooleanProperty	Only set for reaction nodes. If a reaction has this value to true so this reaction is reversible.
subSystem	StringVectorProperty	The list of pathways names in which each node (compound or reaction) is present.
stoichiometry	DoubleProperty	Only for edge. This property contains the stoichiometry attribute of the SBML file. It's the factor applied to the reaction rate to give the rate of change of the species.

Here is a short color algorithm sample to color reaction nodes in blue and species nodes in red.

```
#include <tulip/ForEach.h>
#include <tulip/ColorAlgorithm.h>

using namespace tlp;
using namespace std;

/** Algorithm documentation */
// SpeciesAndReactionColorAlgorihtm will color reaction nodes in blue and species nodes in red. This is just an example to show how to use algorithm with Systrip properties.
/*@{*/

class SpeciesAndReactionColorAlgorihtm:public ColorAlgorithm {
public:

  SpeciesAndReactionColorAlgorihtm(const PropertyContext& context):ColorAlgorithm(context) {
  }

   ~SpeciesAndReactionColorAlgorihtm() {
   }

   bool check(string& errorMsg) {
   //Check if isReaction property exist.
   if(!graph->existProperty("reaction")){
   return false;
   }else{
   errorMsg="Reaction property don't exist, this graph is not a Systrip graph.";
   return true;
   }
   }

  bool run() {
  //Get the reaction property.
  BooleanProperty* isReaction = graph->getProperty<BooleanProperty>("reaction");
  //For each nodes test if the node is a reaction or a species.
  node n;
  forEach(n,graph->getNodes()){
  //The node is a reaction
  if(isReaction->getNodeValue(n)){
    colorResult->setNodeValue(n,Color(0,0,255));
  }else{
    //The node is a species.
    colorResult->setNodeValue(n,Color(255,0,0));
  }
  }
```

```
    return true;
  }
};
/*@}*/

//Register plug-in.
COLORPLUGIN(SpeciesAndReactionColorAlgorihtm, "Species and reaction color", "Authors",  ↩
    "07/07/07", "Comments", "1.0")
```