biotoolsCompose

Manual to the ms-utils/PROPHETS proof-of-concept project

Version 2 of 3 March 16,
by Anna-Lena Lamprecht (anna-lena.lamprecht@lero.ie)

This document contains information on how to install/setup the PROPHETS framework, guides you through a first set of examples, and explains how the domain model was set up.

Installation/Setup

Download the PROPHETS installer (officially: "jABC-Installer-PROPHETS-1.3") from http://ls5-www.cs.tu-dortmund.de/projects/prophets/download.php

Note that jABC and PROPHETS user manuals can be found at https://jabc.cs.tu-dortmund.de/manual/index.php/Main_Page and https://jabc.cs.tu-dortmund.de/manual/index.php/PROPHETS, respectively, but it should be possible to follow this document without studying them beforehand.

Double-click on the downloaded .jar file to start the installation process. Install the jABC to a location of your choice. Note that it requires a recent Java Runtime Environment (JRE). If you do not have one installed, a Java8 JRE can be obtained from: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

Start the jABC by double-clicking on the „jabc" or „Launcher.class", or launch the „jabc.sh" script from the command line. (The latter is the preferred way, as the console will show more details when errors occur.)

Click on "Plugins" in the main menu and check if the PROPHETS plugin is contained there. If not, it needs to be loaded explicitly: Go to "Extras -> Options -> Plugins" and click on the binoculars. This will make the jABC scan the classpath for further plugins. Click "ok" and restart. PROPHETS should now appear in the "Plugins" menu.

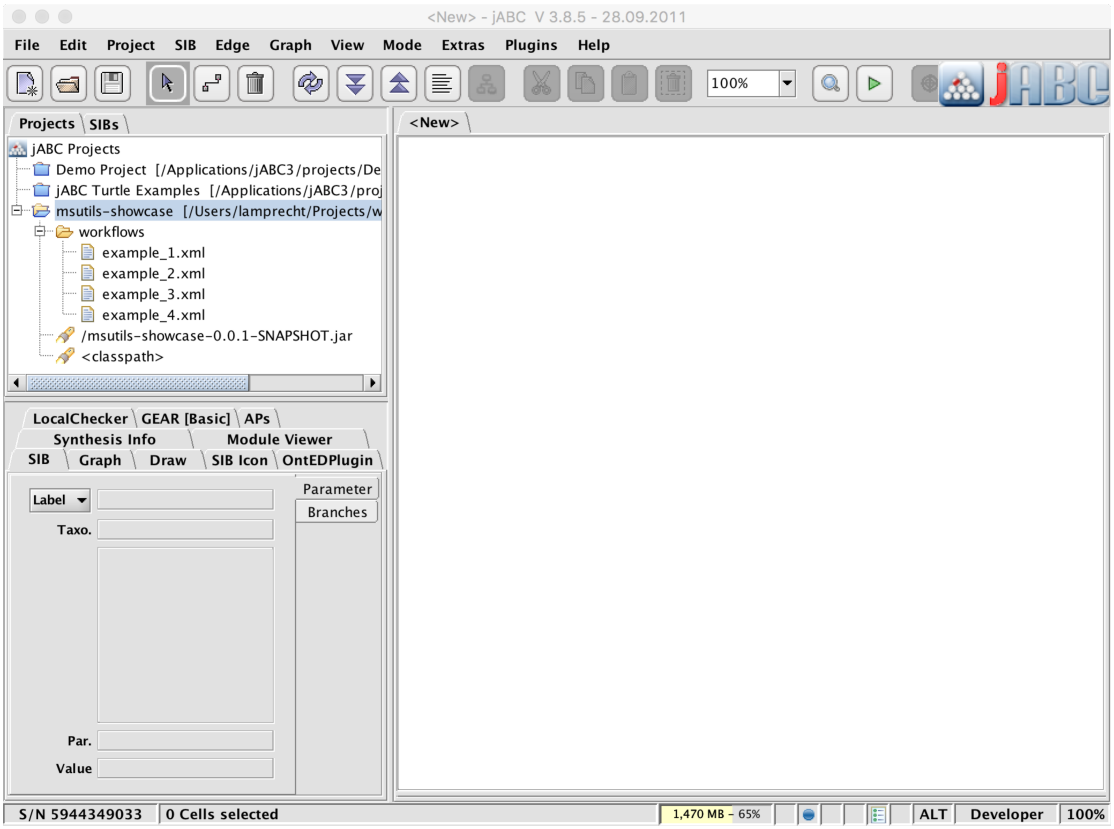
Go to "Plugins -> PROPHETS -> Plugin configuration" and make sure that the parameters are set as follows:

Default Synthesis Process:	globalContext
Debug Mode:	off
Remove Permutations:	no
Goal Constraint:	terminating
Synthesis Algorithm:	mosel
Search Strategy:	bounded

Note: We will explain and change some of them later on, but this configuration makes sure you will be able to see and follow the examples in the next section exactly as described.

Get the demo project from the biotoolsCompsoe Git repository (project “msutils-poc”) and add it as a new project (“Project -> New Project”, then select the project directory). Now your jABC should look more or less like the one in the screenshot below.

If it does, you are ready to continue with the following “guided tour” through the project. If not, contact Anna-Lena for help. ☺



Guided (hands-on) tour through the demo project

Starting point for all example workflows is the following abstract workflow description provided by Magnus in one of the first emails of our discussion:

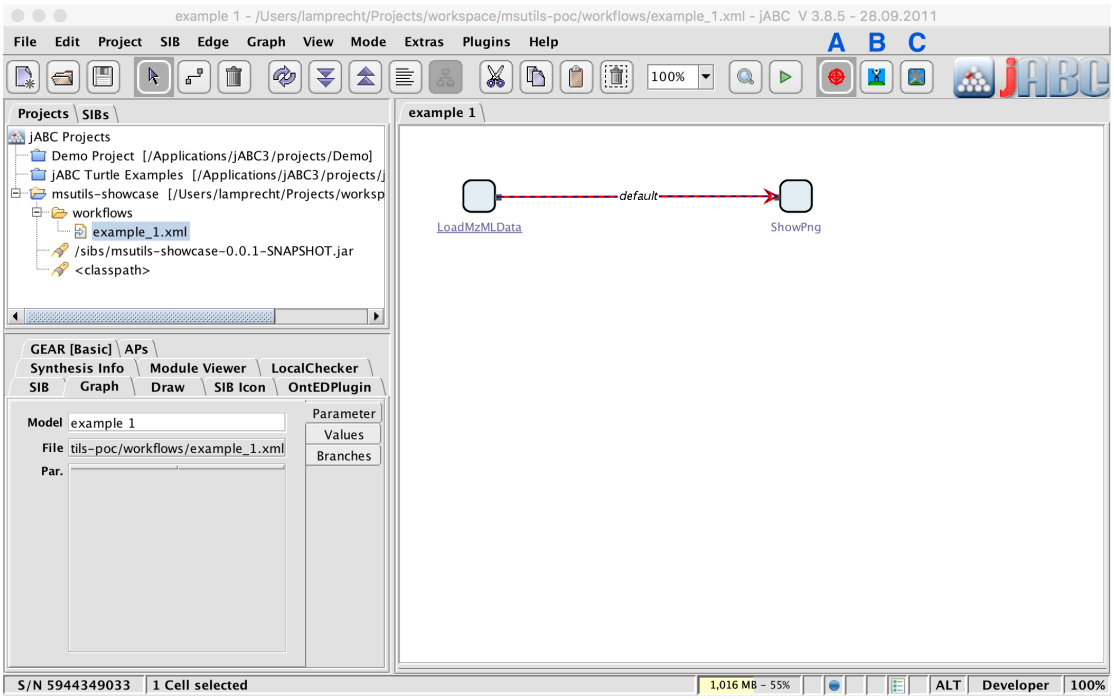
"The analytical chemist does not know anything about these formats, *(s)he only has the raw mass spectrometry data rom the peptides, and wants the amino acid indices out.*"

In PROPHETS terminology, we have a "loose specification" of a workflow here: we know that we have a raw mass spectrometry data set that we can load (first step in the workflow) and what we want to see as a result (last step). But we do not know which tools (and formats) we have to combine, and how, to get there. This is where we ask the synthesis plugin to help us.

The following examples are designed to give an impression what is actually possible, and what working with the tool is like. They raise no claims to completeness at all, but are hopefully suitable to motivate further discussion. ☺

Example 1

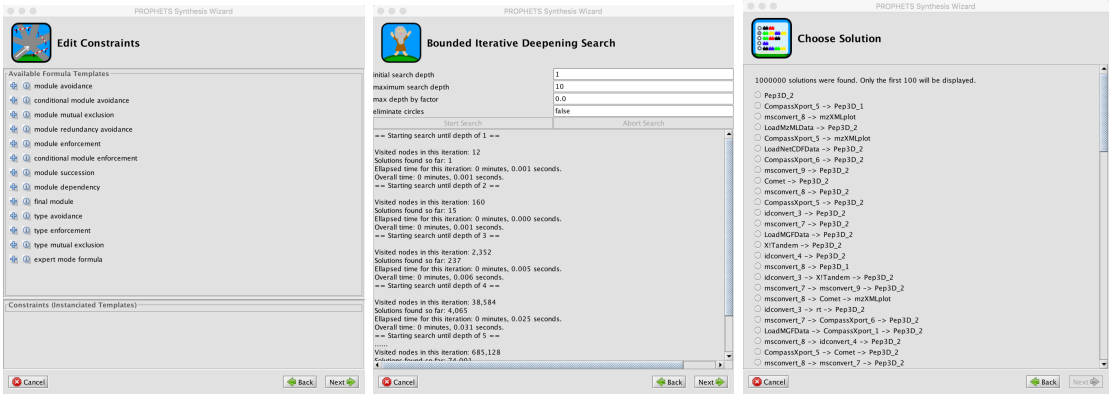
Double-click on "example_1.xml" in the "workflows" subdirectory of the project to open the first example. It should look like in the figure below.



The workflow consists of just two workflow building blocks (or SIBs, as they are called in jABC): "LoadMzMLData", which loads an mzML data set, and "ShowPng", which can display a PNG image (e.g. a chromatogram). The two SIBs are connected by a branch that specifies their order of execution, and that is additionally marked as "loosely specified", which is reflected by its red colouring (there are different ways to mark/unmark a branch as loosely specified, one is

by simply clicking on the button under the “A” in the figure when the branch is selected).

Click on the “Directly synthesize this Branch” button (under the “B” in the figure) to start the synthesis for the loose branch. A series of wizard windows will guide you through the synthesis process.



The first window that is displayed is the constraints editor. We will come back to it later, but just click “Next” for this first synthesis run.
The next window is the search window. Click “Start Search”. The search will be finished after a few seconds (or more precisely aborted because a predefined maximum of solutions is reached). Click “Next” to get to the next window.
The solution chooser window shows the first 100 solutions, ordered by length. The shortest solution to the current synthesis problem is to just insert Pep3D (as it can deal with mzML and directly produces a PNG). Among the two-step solutions are Pep3D and mzXMLplot calls that are preceded by a CompassXport or msconvert conversion step, but also such that contain unnecessary steps like the loading of (additional) data sets. We will see later how to avoid this.

Choose one solution and click “Next”. The next window shows which sequence of SIBs will be inserted into the jABC workflow. This happens by clicking “Next” again. If the selected solution comprises more than one SIB, it will be inserted as a submodel, just to maintain readability.

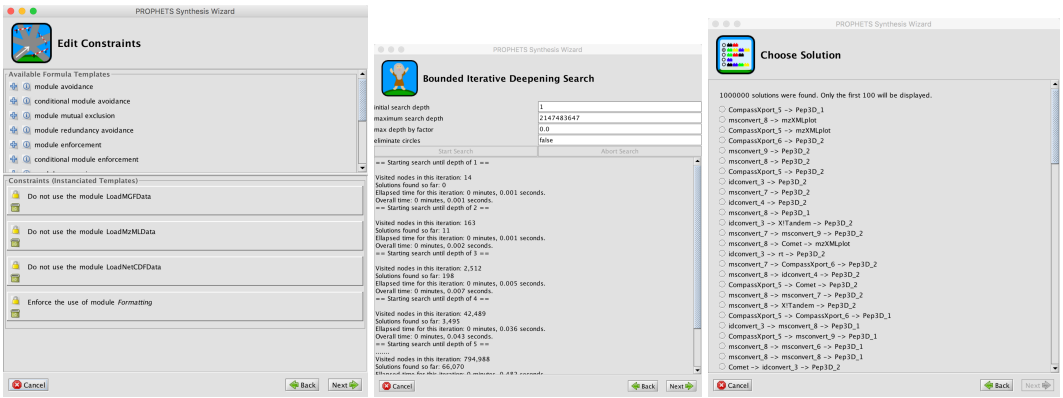
Two more things to note at this point:

1. Most of the tool names have a number attached, e.g. “Pep3D_1” and “Pep3D_2”. This is just to distinguish different input/output combinations that are possible according to the annotations. Pep3D, for instance, is annotated with format_3643 and format_3244 as input format alternatives, so there are two variants of executing it that the synthesis algorithm can assume. We can discuss how to represent this in a nicer and more meaningful way in the future.
2. At the moment none of the SIBs is actually implemented, that is, there is no execution code behind yet, and thus also the workflows will not really do something. But adding calls to the corresponding command line tools etc. should not be a big problem.

Example 2

Now open the workflow “example_2.xml”. It is the same loose specification as in the first example, but with some predefined constraints attached to it. Start the

synthesis for the branch again as before. The same windows will appear, but with slightly different content.



The constraint editor shows four constraints that I have exemplarily defined for this example, based on some observations on the previously obtained solutions. The first three constraints are to exclude the different data loading modules (note: in the terminology of the synthesis algorithm, "modules" are the tools/SIBs/services) from the solutions, as they only introduce additional data and are not necessary for our current purpose. The fourth constraint is to enforce the use of some formatting step on the input (not crucial in the current example, but just as an illustration).
When you run the synthesis now, just as before, you will see that a different set of solutions is obtained: all of them have at least two steps, and comprise a formatting step before the chromatogram visualization, in accordance with the constraints.
The longer solutions often comprise two formatting steps, which does not seem to be very useful, but can also be addressed by defining corresponding constraints if desired.

Note that the constraint editor evaluates the domain model and only allows the use of terms that actually exist in the constraints. Templates are used for their formulation so that no knowledge of the underlying specification logic (SLTL) is required. There is an "expert" mode for directly entering formulas, but the list of templates that is loaded can also easily be extended by adding entries to an XML file.

You are of course free to experiment further with the constraints at this point, but I will continue here with some more aspects and examples.

So far we have used the "globalContext" synthesis process, as configured in the beginning. Its name refers to the data flow mechanism it assumes, namely that all the SIBs in a workflow have access to a shared memory (the "context") into/from which they can write/read data to transfer them from one to another. This is just how workflows in jABC handle the data flow. It can however happen with this configuration that the synthesis includes SIBs in a solution that produce data and put these data into the context for later use, but a later use never happens.
Hence, when synthesizing sequences of services it is sometimes (not always!) more useful to take a "pipelining" perspective, that is, that the output of one SIB is passed only to is successor as input, but nothing else.

Note that at the moment, this perspective seems to make very much sense for our example (only single inputs/outputs annotated, the workflows are pipelines as described above), but this may change when we go further.

To see what happens on the current example when pipelining is applied, change the synthesis process to pipelining ("Plugins -> PROPHETS -> Plugin Configuration", then set the "Default Synthesis Process" to "pipelining") and start the synthesis again as before. You will see that some of the solutions that were obtained when using the "globalContext" synthesis process do not appear here.

Example 3

Now open the workflow "example_3.xml" to take a look at what happens when we search for a workflow that computes amino acid indices (in textual format). Start the synthesis as before and you will see that loads of solutions are found. Apparently, however, the goal description as "textual format" (format_2330 in EDAM) is so general that not only the amino acid indices computed by the tool set fulfil this specification, but also some idconvert output, for example.
There are different possibilities how this can be resolved, including the refinement of the tool annotations to be more precise. One less invasive way of getting more adequate solution is to define a constraint that says more clearly what we want to have.
Start the synthesis for this example again. In the constraints editor, add a "final module" constraint (just click on the + left of the "final module" template in the list of available templates). From the drop-down menu within the appearing template sentence, select "Retention_times_predcition" and click on the padlock symbol on the left to finish editing of the template. In the next window, start the search and wait for it to finish. From the outputs in the search window, you can already see that no one-step solution to this specification exists. Click "Next" and look at the solutions. They all have rt4 as a last step to fulfil the newly added constraint. Still, there are things in the solutions that do not make too much sense, but like for the previous examples, constraints can be defined here to resolve them. For example, add constraints to avoid the Load* tools (as above) and to forbid the use of Formatting more than once (constraint template "module redundancy avoidance") and see what happens.

Example 4

As a last example for the moment, let's have a look at the workflow in "example_4.xml". The loosely specified workflow here also starts with loading an mzML data set, but does not define a particular desired output. This is useful to explore what (else) can be done with the input data set given the available tools. Constraints are used to exclude further data loading steps from the solutions, but not more. In the PROPHETS configuration, set the synthesis process to "pipelining" and the goal constraint to "simple" ("terminating" would not do anything here, because "any solution" is already available from the first step, and would cause the synthesis to stop the search immediately).

Again, numerous solutions are found, and the 100 displayed by the solution chooser are not longer than 3 steps. With defining more constraints (e.g. forbidding repetitions of particular tools or groups of tools, or explicitly stating

what is actually wanted) the amount of solutions can be reduced and better aligned to what is intended. Feel free to play with that. ☺

The solution chooser will only show the first 100 solutions (assuming that no one wants to look at more, especially in the current textual representation). For visualizing the possible solutions in a more compact way, there is a "hidden" feature for creating an automaton representation in .dot format from the solutions. Currently it can only be switched on by changing the prophets.solutionStore setting in the PROPHETS.properties file (located in the .JavaABC directory, which should be located in your home directory) to "automaton", so that during the synthesis process a .dot file is written to the temp folder (the filename is show on the console). The synthesis process is slower with this setting, so this is not a good standard configuration. The .dot file for the workflow of example_4.xml with the settings as described above is available in the project directory and can be opened with the DOTTY tool (http://graphviz.org), and I have also added .ps and .pdf versions of what DOTTY displays.
Note that the synthesis will always stop when 1.000.000 solutions have been found (assuming that this is really more than anyone could want to handle), and in most cases there are infinitely many solutions possible in principle.

How the domain setup works

This section now takes a look "under the hood", explaining what the PROPHETS domain model comprises and how it was set up (semi-) automatically. The corresponding files are hidden from the user (you didn't see any of them while you went through the previous section), but they are crucial for the synthesis to work correctly. Concretely, these files are:

- The .jar file(s) containing the SIBs for the project. In our case, it is only one file, namely "msutils-showcase-0.0.2-SNAPSHOT.jar" that is contained in the project directory and has been added as a SIB path.
- The moduleTayonomy.owl and typeTaxonomy.owl files in the "prophets" subdirectory of the project that contain taxonomies for defining and classifying the modules and types that are used in the domain model, respectively.
- The modules.xml file in the "prophets" subdirectory of the project, which contains the description of the modules for the synthesis (module name, corresponding SIB, branch, input and output types).
- The templates.xml file in the "prophets" subdirectory of the project that contains the templates that are provided by the constraints editor. It is currently just the default version, but additional templates to be provided by the constraint editor could simply be defined here.

The .jar file has been created manually and comprises SIBs corresponding to the tools selected for the moment. As mentioned above, they have not been fully implemented yet, but they are sufficient for experimenting with the synthesis as such and their implementation (basically adding code that calls the respective command line tools, and corresponding parameters) should be a more or less straightforward business. Technically, a SIB is simply a Java class that implements a couple of specific interfaces. Their code is available in the biotoolsCompsoe Git repository (project "msutils-poc-domainsetup", package "biotools.biotoolCompose.poc.sib").

In the package "biotools.biotoolCompose.poc.domainSetup" this project comprises a Java script that automatically creates the taxonomies and the modules.xml file from EDAM and from a .csv file derived from the ms-utils annotation spreadsheet. I try to explain the essential aspects of what is happening there in the following, if you have further questions just ask, or see the code for full details.

In the first phase, the EDAM ontology is imported. That is, the "Operations" class from EDAM and the tree of subclasses below it are turned into a PROPHETS-compatible module taxonomy, and the "Format" class and its subclasses are turned into a PROPHETS-compatible type taxonomy. There is some legacy code involved here and the implementation is not as elegant as I would like it, but in principle this process is straightforward and the outcome serves the current purpose.

Next, the .csv file is processed to create the modules.xml and extend the taxonomies accordingly. For each tool and each combination of input/output formats that is possible according to the annotation in the file, a module is

created and added to the modules.xml file. The module is also added to the module taxonomy as subclass of the operations that are given in the annotations file. As for the input/output type descriptions only terms from the type taxonomy are used, nothing needs to be updated here.

If you have a look at the annotations.csv file, you will see that it is still very similar to the original annotation spreadsheet. Three changes were made:

1. The tools that are currently not considered in the example have been removed.
2. Tools corresponding to the auxiliary SIBs for loading and showing data have been added, simply to include them in the automatic domain setup without introducing further files.
3. The "Comment" column has been filled with text for all tools, just to make sure that all rows have the same number of columns when the table is saved in .csv format, since this simplifies its processing.

This is not ultimately stable, but I think for the moment it is an easy way to keep the domain information for our experimentation with the synthesis aligned with your ongoing annotation efforts.

Finally, the taxonomies are post-processed to remove all terms that are not used in the domain model. Or expressed the other way round: only the terms used in the modules.xml and all their superclasses on the way to the root are kept. This taxonomy pruning is not necessary, but makes the synthesis process a bit faster and the domain model easier to use, since the user does not see loads of irrelevant terms, for instance when filling a constraint template.

Finally a technical note, only relevant when you want to run the script or add/change SIBs and create a new .jar file: It is a Maven project, and some of the dependencies are downloaded from a server at TU Dortmund where you will not have access. There is an easy workaround, however: Simply add the .jar files from the PROPHETS installation ("lib" and "plugins" directories) to the classpath of the project, and it should contain everything that is needed.