Refactoring of the CAPRI GUI and integration of new functionalities

- Wolfgang Britz, November 2008 –

Motivation

The basic concept of the current Graphical User Interface (GUI) of CAPRI dates back to 2004 and developed out of a trial prototype. Albeit being quite successful in its application, the maintenance and especially further expansion of the code base becomes more and more cumbersome. One major reason is a strong intermingling of the GUI handling and the business objects in the code.

The aim of the refactoring is to generate a technically clearer structure while integrating newer features as the meta data facility, batch execution, HTML based documentation of the GAMS code à la javadoc and links to SVN. In order to do so, major parts of the code base of the GUI needed reprogramming, building on the concept developed for the batch execution facility.

Basic class layout

CAPRI tasks as business objects

A core concept in the new layout is a business object called *AgpTask*. Technically defined as an interface, such an object represents a work task in the overall CAPRI system such a run of CAPREG to build the regional data base. The interface requires getters and setters for properties such as *baseYear*, *simYear or MemberStates*. The setters can be accessed either by a GUI interface or by the batch execution facility, formally by class implementing the interface *AgpTaskHandler*.

Most tasks are GAMS executable tasks according to their *isGams* property. These tasks also provide access to the name of the related GAMS program via *getGamsProgramName*. Each of these tasks has also a method called *generateIncludeFile* which generates the specific GAMS include file for that task.

The objects also know about the main GDX file they are generating via *getGdxResultFiles*. Related to that, they allow setting the logical names of the data dimension in the result data set via *setDimNames* and *setXMLTablesDims*.

Once the properties of a task had been defined, their logical consistency can be checked by invoking the method *checkSettings*. Check settings returns a string with a description of the first error encountered.

That layout eases dramatically the update process of CAPRI. Definition of new tasks or changes to existing ones will generally not require changes in the GUI, but simply either implementing a new object with the required methods or updating an existing one.

Execution of tasks via a GamsStarter and GamsThread

Execution of tasks with the property *isGams* is handled by a *GamsStarter* object. An instance of *GamsStarter* let the task write out the necessary include file(s) to generate a specific instance of the specific task (a simulation run for a specific scenario, simulation year, with the market model switched on or off …). A *GamsStarter* also knows about the working directory or other specific GAMS settings as the scratch directory. It may generate a pipe for the GAMS output to the console to show it in a GUI.

An *AgpTask* can be executed by a *GamsStarter* who will create then a *GamsThread*. A *GamsThread* extends the *SwingWorker* interface of Java so that it may communicate with the

normal event queue of JVM. A *GamsThread* can be gracefully terminated by sending a a SIGNT signal to the GAMS process. That will let the GAMS execution stop at specific point determined by the GAMS engine itself and start the finalisation handling of GAMS as removal of intermediate files and directories.

Changes in the exploitation tools

The refactoring did not attack reprogramming the exploitation tools (tables, graphs and maps). Here, a clear design proposal needs first to be developed before the actual coding may start, aiming at a separation of the data model, user based filtering (empty lines and columns, filters based on values in specific columns), layout and filtering loaded from the table definitions and the actual view (table, graphics, maps), pivot, colors, fonts etc.. Several attempts to clean up of the code already ended rather unsatisfactory.

In the very first version of the table tool without any pre-defined views, the code was rather strictly separated from the GUI for the working steps. However, when new features were added to the GUI such as progress bars, or hiding part of the GUI controls when the exploitation tools were active, properties and methods of the GUI were accessed from within the exploitation tools. As so often, missing resources for a serious refactoring and the rule of "never change a running system" led to serious design flaws. A lot of the GUI controls were accessed by inner classes and then defined static, and together with some problems ended as public static objects, often directly accessed from anywhere in the exploitation tools. Removing these snippets in the code is one major reason to clean up the GUI, as it is the first step towards a new implementation of the exploitation tools.

In order to allow for a limited time co-existence between the old and the new GUI, interface classes with these functionalities had been generated and implemented in both versions.

Changes in user interface

An important change is that the separation of execution and exploitation work steps had been removed. So far, there was typically for each task in an execution work step (build national data base) a matching exploitation task.

Instead, the different tasks related to runs of GAMS code can now be exploited directly. A disadvantage of that layout is the fact that controls may be accessible which do not impact on the results exploited. The main advantage is a clearer view by the user which runs generate which results. Even more important is the underlying clarity of the basic concept: each

executable task generates one major set of results which is stored in a GDX file. Along with the results, meta data on the data used by the task and the task itself are stored in the GDX file.

The layout required so far solely one conceptual change: as CAPREG is generating (still) in one run time series data and the three year average, a “clone” of the CAPREG task allows accessing the time series. However, it is anyhow planned to separate CAPREG into these two tasks.

An exemption to the rule above is the exploitation of scenario results. Here, results of several runs need to be compared in parallel. It is the only “exploitation” only work step related to the production work flow.

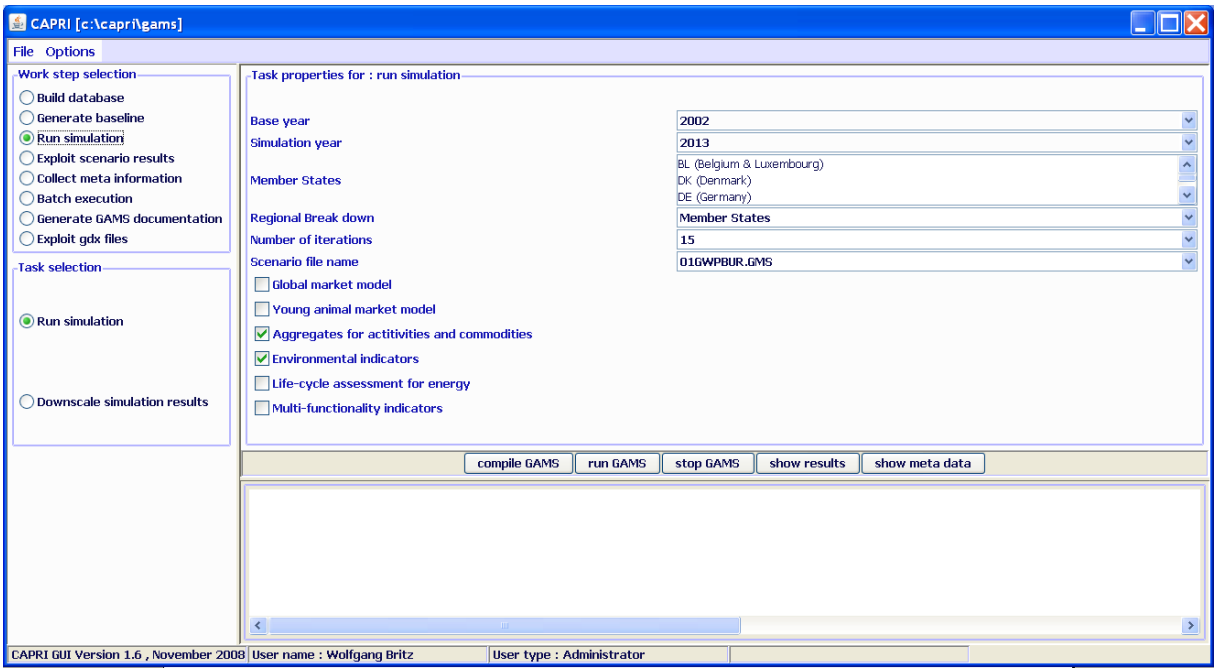
As discussed below in more detail, three new work steps had been added which are not directly linked to the production work flow:

1. *Collection of meta information* from the different work steps by accessing the meta data information stored in the GDX files generated by the different work steps.
2. *Batch execution* of test suits or predefined production runs.
3. *Generation of HTML based documentation of the GAMS code*.

Finally, as in the old GUI, symbols from GDX files may be viewed based on the exploitation tools.

A second difference to the existing GUI is the fact that all run specific settings are now found in one panel. Again, it eases code development and maintenance. But it is also thought to render the GUI more uniform. The reader is reminded that it is planned to generated “reporters” which can be called on results sets. Once these are implemented, the steering possibilities for “run simulation” will become smaller, as reports will be started separately.

An example can be found below. All controls for “run simulation” are now moved to the upper right corner window. As can be seen from the screen shot, the basic layout of the GUI along with colouring was not changed compared to the old GUI to reduce learning costs. Users familiar with the old GUI should not face any difficulties in steering the new one as the name of the controls and their functionality was not changed.



Integration of META data information

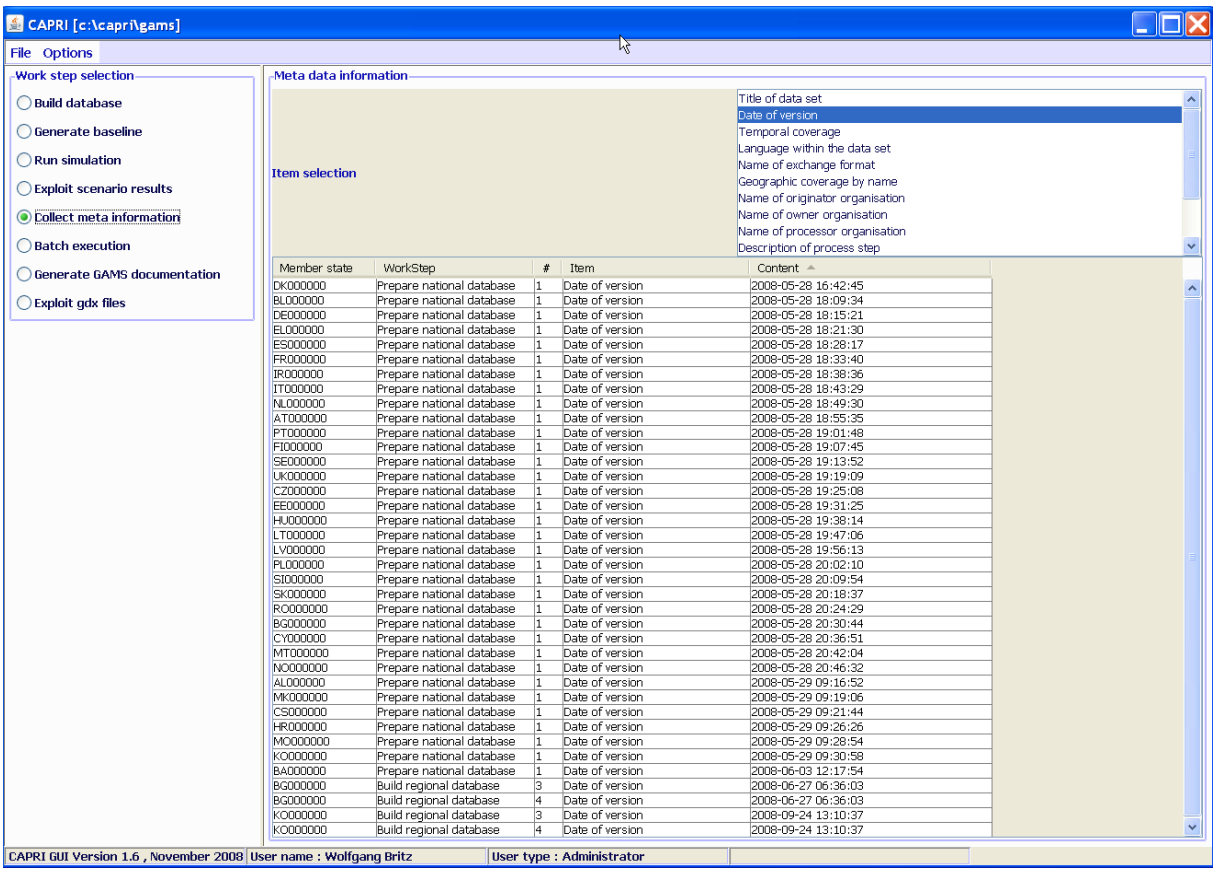
As already in the old GUI, the meta data from any work step can be loaded with the “show meta data” button. That view allows a detailed control of the different data sources used by the different work steps.

The tasks store basically three different types of meta information. Firstly, most data sets read by a work step carry by now appropriate meta data which is added to the meta information for the task. Secondly, when a task is started, its GAMS include carries meta information about the current run (the user, data and time, settings etc.). And thirdly, meta information from results sets of earlier runs used by the current run are added.

A new feature still in the development and test phase is the “collect meta information” view. As seen below. It lists for each task two main meta data per Member State: the user who started the task, and the data and time of execution. Each task has a sequence number (#), and the underlying class *AgpCollectMeta* will show a line a task in red if the current version on disk for a task with a smaller sequence number is younger. As the META data and the results are stored in the same GDX file, shipping files between computers e.g. via the SVN server can never separate the meta information and the actual data, and the “Collect meta information” task will thus use always the actual information from the data files. The actual layout of the table will certainly change, and most probably based on a revised implementation of the normal exploitation tools.

As a first step in direction of assisting users in checking the logical consistency of the data in use, the table will print an entry – the combination of a Member State and a work step - in red if the underlying results from a previous work step are younger. At a later stage, that must be expanded to cover also data sources commonly used by different work steps.

For details on the integration of meta data in CAPRI see the technical document “Integrating meta information in the CAPRI production chain” to be found on the Capri web page under technical documents.



Integration of the batch execution facility

The batch execution facility is a tool which:

- Allows executing many different CAPRI task after each other without requiring user input.
- Reports the settings used, any errors and GAMS result codes in a HTML page from which they may queried at a later time.
- Ensures that each new run generates its own listing file, which can be opened from the HTML page.

- Allows storing the output of the different runs in a separate directory, while reading input from unchanged result directories.

The purpose of the batch execution facility is therefore at least twofold. At the one hand, it allows to set up test suits for the CAPRI GAMS code such as checking for compilation without errors for all tasks and different settings such as with and without market parts etc.. Secondly, production runs of e.g. different scenarios can be started automatically. It is planned to add timer facilities to the batch execution so that the GUI will start a suite of runs at a pre-scheduled time. Along with the planned functionalities to compare in a more or less automated way differences in results between versions, the batch facility is one important step towards quality control.

For details on the batch execution facility see the technical document “Batch execution of CAPRI tasks” to be found on the Capri web page under technical documents.

So far, the batch execution facility was only used for tests, and started “blindly” in the console. Now, the batch execution facility can be started from the GUI, and the console output of GAMS is shown in the GUI.

The steering of the batch execution facility is very simple: (1) selecting the file with the batch command, and (2) hitting the start button. A run batch job may be cancelled such that the running GAMS code is terminating on its own, or by sending a SIGINT signal to the current GAMS job, requesting its termination.

The batch execution facility is linked to the HTML based code documentation a la javadoc: the generation of the necessary input files (reference and expand files from GAMS) can be switched on with a check box in the GUI. Equally, the directory where these files are generated can be determined.

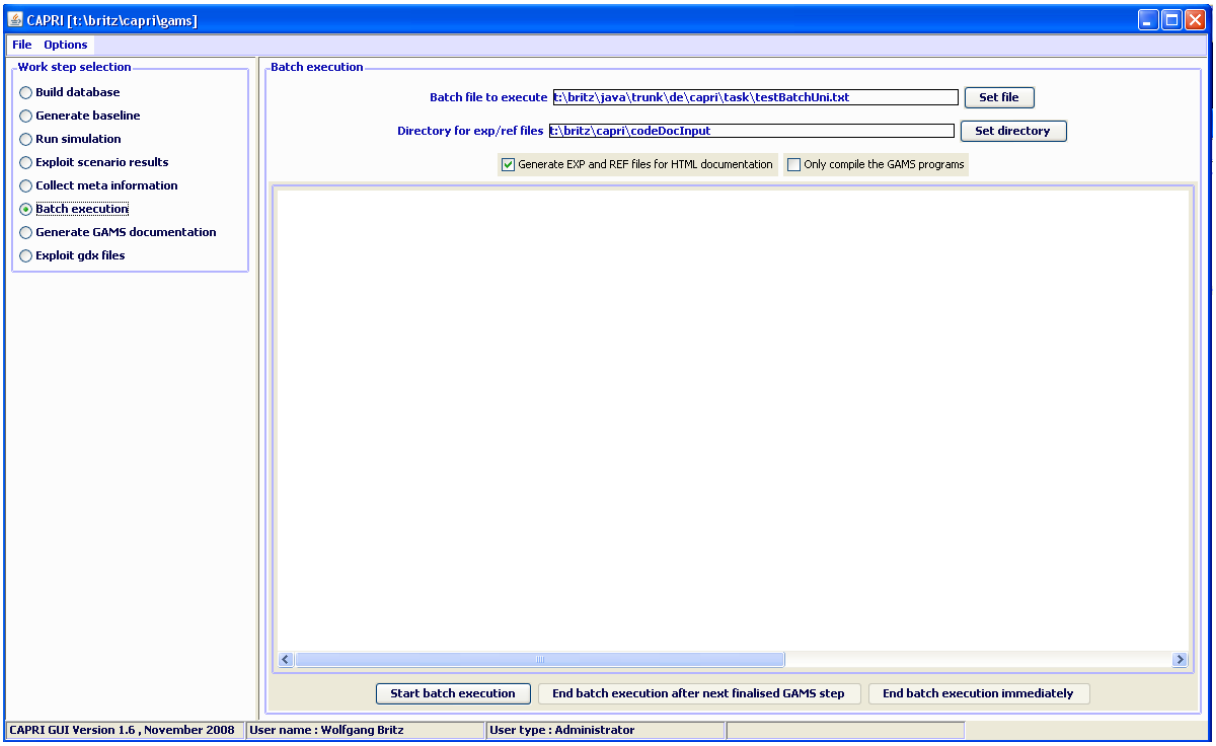
There is still some coding necessary as currently the name of the files are equivalent to the GAMS program called. As long as e.g. policy shifts, baseline calibration and a simulation run are all based on CAPMOD, the last CAPMOD run will be documented.

Integration of the HTML based code documentation

Another feature developed and not yet integrated in the old GUI was the HTML based GAMS code documentation. It builds on information provided by GAMS via specific files (reference and expand files) which can be generated by the GAMS compiler. The steering so far is rather

straightforward. The user selects the directory where the expand and reference files are to be found, and the GUI loads all files found. Now, the user selects the programs to document and start the HTML based documentation generation.

In order to test a batch execution file, the “only compile GAMS programs” check box can be activated. In that case, the batch execution facility will not execute the GAMS programs which should prevent overwriting existing results.



For details on the code documentation facility see the technical document “Javadoc like technical documentation for CAPRI” to be found on the Capri web page under technical documents.

Next steps

The final release of the new version and removal of the old one is planned in 2009 after the new interface has been thoroughly tested. The old code will be removed from the code base, so that any remaining references to static instances of Capri in the exploitation part can be removed. A similar refactoring is certainly necessary for the exploitation part of CAPRI.