TradeRES
New Markets Design & Models for 100% Renewable Power Systems

User Guide

AMIRIS

Agent-based Market model for the Investigation of Renewable and Integrated energy Systems

Overview

AMIRIS is a next-generation tool to dissect the complex questions with respect to future energy markets, their market design, and energy-related policy instruments. The model computes electricity prices based on the simulation of strategic bidding behavior of prototyped market actors. This enables AMIRIS to not only consider marginal prices but also support instruments and uncertainties. Figure 1 shows agents and their associated flows of information, energy, and money modeled in AMIRIS.

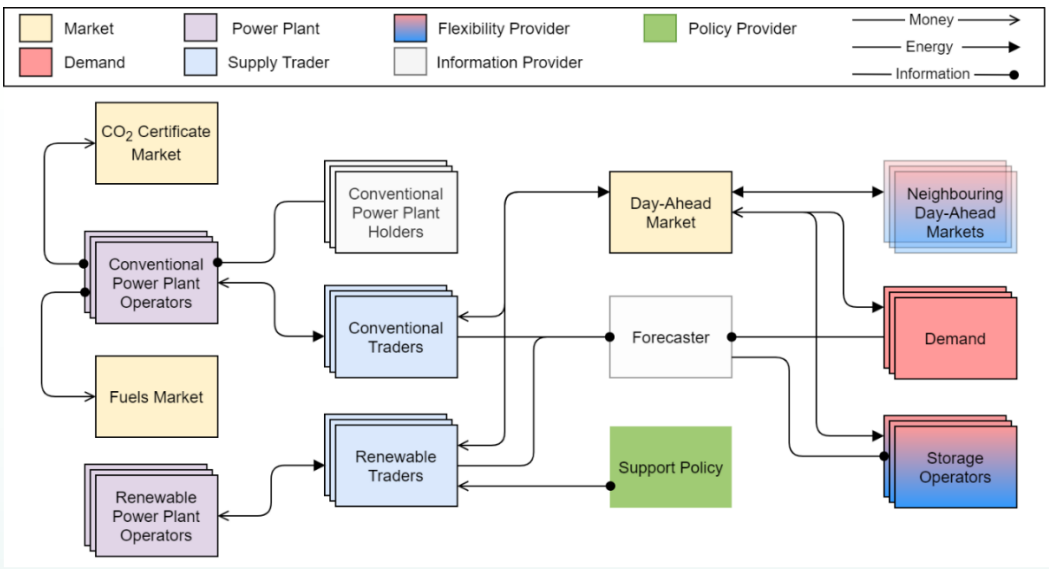

Figure 1 Overview of agents and their interactions in AMIRIS (open version)

Actors are represented as agents and can be roughly divided into six classes: Power plant operators, traders, marketplaces, policies, demand, and flexibility option facilities. Power plant operators provide generation capacities to traders, but do not trade on the markets themselves in the model. Bidding and operation decisions are conducted by traders in pursuit of, e.g., profit maximization strategies. Marketplaces serve as trading platforms and determine prices. Policies define a regulatory framework, which impacts the decisions of other agents. Demand agents as well as flexibility option facilities, e.g., storage facilities, trade directly in the electricity market.

Inputs

AMIRIS is configured via human-readable YAML files: “scenario.yaml” and “fameSetup.yaml”. The latter is covered in the section “How to run AMIRIS”. The scenario file is split into several sections:

The “Schema”-section specifies which types of agents exist, which attributes they have, which they require and how they can interact with other agents. This section must not be changed. Typically, a separate schema file is provided to describe simulation capabilities and requirements. The special YAML loader of FAME-Io https://gitlab.com/fame-framework/fame-io allows to split and reuse YAML files via the “!include“-command.

The “Agents”-section defines which agents are to be created in the simulation and how they are parameterized. Each agent requires at least a type and unique id. Additional attributes might be required, depending on its type. Attributes can be a single value, a list of values, an externally specified time series in CSV format, a group of sub-attributes or even a list of grouped sub-attributes. The type of each attribute is specified in the Schema section. The full list of attributes cannot be stated in this document. Instead, please refer to the up-to-date pages at the AMIRIS-Wiki https://gitlab.com/dlr-ve/esy/amiris/amiris/-/wikis/Classes/Classes. There, a comprehensive list of all attributes and associated configuration options can be found for all agent types.

The “Contracts”-section configures the interactions between the agents. Each contract comprises a sender and receiver agent (identified by ID), a product type, as well as an initial execution time and execution interval (in seconds). Usually, changes to contracts are only required if agents are added to or deleted from the configuration. For this, groups of agents are defined that allow to add and remove agents conveniently with a change at a single point of the configuration. Please see the AMIRIS-Wiki for a list of all available products per agent type. Similar to the “Schema” section, “Contracts” are often extracted to separate files to keep the scenario file tidied up. FAME-Io’s “!include” command can be used to load a multitude of other YAML files. See the AMIRIS-Examples project https://gitlab.com/dlr-ve/esy/amiris/examples for several examples of techniques to keep the configuration files neat and organized.

Contract configuration for AMIRIS is not trivial, since almost all actions within AMIRIS are controlled via contracts. Thus, one needs to know what actions create which data, which input is required by what action and which agent can provide such input. Therefore, instead of starting from scratch, please refer to the AMIRIS-Examples project to see several working examples of agents and related contracts.

Once the configuration is completed, “FAME-Io”, a Python tool, is used to convert configuration files into a single binary input file for AMIRIS. Please see the FAME-Io documentation for further instructions on its installation, execution, and command-line options.

Outputs

Each execution of AMIRIS creates a singular output file. Name and path to that file can be controlled via the “fameSetup.yaml” (see next section). The binary output file (protobuf) needs to be converted to human-readable form for interpretation. FAME-Io provides a script to perform that task, which will create a folder with CSV files: one file per agent type in the simulation. Figure 2 provides an example output in that format.

```
AgentId;TimeStep;TotalAwardedPowerInMW;ElectricityPriceInEURperMWH;DispatchSystemCostInEUR
1;599184000;52799.0;0.0;382017.32681056164
1;599187600;42766.0;0.0;382017.32681056164
1;599191200;50965.0;0.0;382017.32681056164
1;599194800;50546.0;0.0;382017.32681056164
1;599198400;39285.0;0.0;382017.32681056164
1;599202000;51366.0;0.0;382017.32681056164
```

Figure 2 Sample output of an agent of type “EnergyExchange” in CSV format

Each created output file features the columns “AgentId” and “TimeStep” at least. The first column refers to the ID of the agent as specified in the “scenario.yaml” file (see previous section). All outputs from agents of the same type are combined in the same file, although sorted by “AgentId”. The next column defines the simulation time at which the output was made by that agent. Please see https://gitlab.com/fame-framework/wiki/-/wikis/TimeStamp for a detailed description how FAME measures time. FAME-Io offers a function to convert time steps to time stamps – when using Spine toolbox (see next section), this is done automatically.

Depending on the type of agent, additional columns exist. Typically, the header row tells what value is depicted – including the unit in its name. Note that not all output columns are used in each time step, necessarily. Figure 3 provides an example for that case. There, outputs alternate between specifying offered and awarded power, which occurs at different times within the simulation.

```
AgentId;TimeStep;OfferedPowerInMW;AwardedPowerInMWH;ReceivedMoneyInEUR;CostsInEUR
500;599183998;9135.36;;;
500;599184002;;9135.36;0.0;55285.4799328693
500;599187598;9135.36;;;
500;599187602;;9135.36;0.0;55285.4799328693
500;599191198;9135.36;;;
500;599191202;;9135.36;0.0;55285.4799328693
500;599194798;9135.36;;;
500;599194802;;9135.36;0.0;55285.4799328693
```

Figure 3 Sample output of an agent of type “ConventionalPlantOperator”

A full list of all outputs cannot be given here. Please refer to the AMIRIS-Wiki to learn about all agent types and their outputs. You may also check the source code and inspect what is actually done to create the output. Simply search for uses of the "store(<ColumnName>, <Value>)" method in the particular class for the agent type you are interested in.

How to run it

AMIRIS is based on FAME, the "open Framework for distributed Agent-based Modelling of Energy systems" (see https://gitlab.com/fame-framework). Thus, it requires a Java Development Kit (version 8 or above), a Python installation (version 3.7 or above) and Apache Maven. AMIRIS can be either run standalone or within the Spine toolbox. Therein, all individual steps of input peparation, running and output conversion are joined into a single workflow. An experimental workflow to run AMIRIS within the Spine toolbox is availlable at https://github.com/TradeRES/toolbox-amiris-demo. However, usability and documentation is to be enhanced in the near future. Please see the AMIRIS Readme file https://gitlab.com/dlr-ve/esy/amiris/amiris/-/blob/main/README.md for a description on installing and running AMIRIS without Spine toolbox. The Spine toolbox workflow requires a packaged Java ARchive (JAR) file of AMIRIS including all dependencies. It can be easily obtained using Maven – please follow instructions in the AMIRIS Readme.

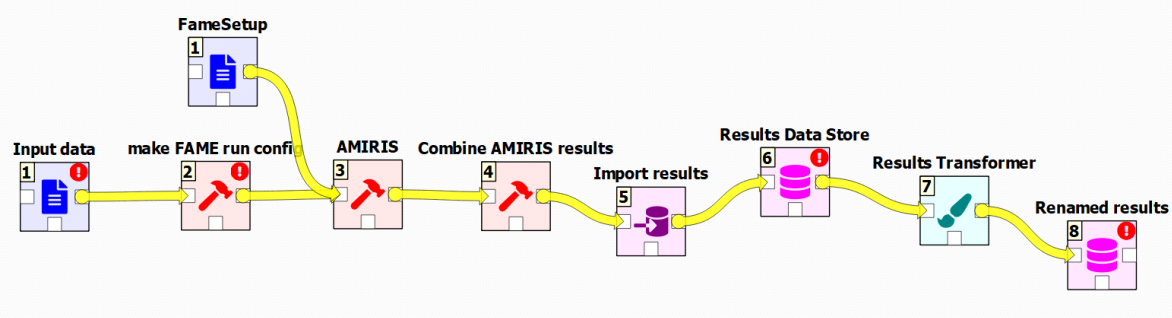

Figure 4 Workflow executing AMIRIS in Spine toolbox

Figure 4 illustrates the workflow steps in the Spine toolbox: two files (1) need to be provided to the workflow: the *scenario* definition and the *fameSetup* – both in YAML-format. Please see the section Inputs for a description of the scenario file. The workflow automatically calls FAME-Io (2) to translate the scenario into a single binary input file in protobuf format to be used by AMIRIS. When AMIRIS is run (3), the *fameSetup.yaml* file is read by FAME-Core. It defines file output parameters (see Table 1). It is best not to change the provided file. After AMIRIS is run, the result file is read (4), extracted into .csv files and imported (5) into the local SQL database (6). To comply with the TradeRES naming standards and assigning of

time stamps to the results, these data are then transformed (7) and saved into another database section (8).

Table 1 Parameters in fameSetup.yaml

Parameter	Description
outputPath	Relative or absolute path to create the output file at
outputFilePrefix	Name of the output file
outputFileTimeStamp	True on default; if true, a time stamp is prepended to the output file
agentPackages	List of Java package names that contain classes derived from “Agent”
messagePackages	List of Java package names that contain classes derived from “DataItem”
portablePackages	List of Java package names that contain classes derived from “Portable”

TradeRES

New Markets Design & Models for 100% Renewable Power Systems

Find out more

AMIRIS Home https://dlr-ve.gitlab.io/esy/amiris/home/

AMIRIS@Gitlab https://gitlab.com/dlr-ve/esy/amiris/amiris

AMIRIS@openMod https://forum.openmod.org/tag/amiris

FAME@Gitlab https://gitlab.com/fame-framework

FAME-Core@Maven https://mvnrepository.com/artifact/de.dlr.gitlab.fame/core

FAME-Io@PyPI https://pypi.org/project/fameio/

Nitsch, F. and Schimeczek, C. and Bertsch, V. (2021) “Back-testing the agent-based model AMIRIS for the Austrian day-ahead electricity market”. *Working paper*. doi: 10.5281/zenodo.5726738

Nitsch, F. and Deissenroth-Uhrig, M. and Schimeczek, C. and Bertsch, V. (2021) “Economic evaluation of battery storage systems bidding on day-ahead and automatic frequency restoration reserves markets”. *Applied Energy* (298). Elsevier. doi: 10.1016/j.apenergy.2021.117267.

Laura Torralba-Díaz et al. (2020) “Identification of the Efficiency Gap by Coupling a Fundamental Electricity Market Model and an Agent-Based Simulation Model”. *Energies*. Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/en13153920.

Frey, U. and Klein, M. and Nienhaus, K. and Schimeczek, C. (2020) “Self-Reinforcing Electricity Price Dynamics under the Variable Market Premium Scheme”. *Energies*. Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/en13205350.

Deissenroth, M. and Klein, M. and Nienhaus, K. and Reeg, M. (2017) “Assessing the Plurality of Actors and Policy Interactions: Agent-Based Modelling of Renewable Energy Market Integration”. *Complexity*, 2017, 7494313:1-7494313:24, doi: 10.1155/2017/7494313.

Info

The TradeRES project will develop and test innovative electricity market designs that can meet society’s needs of a (near) 100% renewable power system. The market design will be tested in a sophisticated simulation environment in which real-world characteristics such as actors’ limited foresight into the future and risk aversion are included.

Start date
1 February 2020

End date
31 January 2024

Overall budget: € 3 988 713,75

https://traderes.eu
info@TradeRES.eu

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 864276