Technical Report

Project Franklin: An Australian Electricity Market Simulator

Authors:  
Luke Horvat  
Michael Jensen

Supervisors:  
Bao Vo  
Clinton Woodward

November 20, 2011
Abstract

In order to investigate the effects of various events and changes in the Australia’s National Electricity Market (NEM), a system has been proposed to simulate the interactions between entities operating within the market by using a multi-agent approach. This system produces data on the allocation of generating units in the electrical system based on the sale price offered by each unit, but can be extended in future works to more comprehensively model the wholesale electricity market.
## Contents

1 Introduction ........................................ 1  
1.1 Purpose ......................................... 1  
1.2 Background ...................................... 1  
  1.2.1 The Australian Electricity Market ............ 1  
  1.2.2 Market Power .................................. 2  
1.3 Scope of Research ................................ 2  
1.4 Significance ..................................... 2  
1.5 Tool Requirements ............................... 3  

2 Proposal ............................................ 3  
2.1 Software Architecture ............................ 3  
  2.1.1 Introduction .................................. 3  
  2.1.2 Code Style .................................... 4  
  2.1.3 Software Modules ............................... 4  
  2.1.4 Typical Usage Flow ............................ 5  
  2.1.5 Classes ....................................... 5  
  2.1.6 Experiment Configuration ...................... 6  
2.2 Agents ........................................... 6  
  2.2.1 Architecture .................................. 6  
  2.2.2 Communications ............................... 6  
  2.2.3 Initialisation .................................. 8  
2.3 Customisable Simulation Configurations ............ 9  
2.4 Events ........................................... 10  
2.5 Data Monitors .................................... 11  
2.6 Data Providers ................................... 11  

3 Discussion ......................................... 11  
3.1 Existing Systems .................................. 11  
3.2 Limitations ...................................... 11  
3.3 Future Work ..................................... 12  
  3.3.1 Projects ...................................... 12  
  3.3.2 Framework .................................... 12  

4 Conclusion ........................................ 12  

5 References ........................................ 13
1 Introduction

1.1 Purpose

The purpose of this document is to detail the design of the Franklin Electricity Market Simulator (Franklin) and the reasons for its development. This is a technical document that describes how Franklin was designed and built, along with justifications for the design decisions made during development.

1.2 Background

1.2.1 The Australian Electricity Market

In order to ensure the fair distribution of electricity generation business, as well as ensuring that a sufficient quantity of electricity is generated to meet the population’s demand, the National Electricity Market (NEM) of Australia was formed in 1998. The NEM was a consolidation of all of south east Australias gas and electricity service providers, following changes in the regulation and privatisation of state owned assets. Until 2009, the parts of the NEM were controlled by different companies including NEMMCO, VENCorp, ESIPC, REMCO, GMC and GMRO. The Australian Electricity Market Operator (AEMO) was formed in 2009 and amalgamated all existing energy markets in Australia. AEMO’s was one of the first entities in the world to operate a transparent and highly competitive market, controlled by strong governance. Figure 1 shows an example graph of electricity price within a single trading interval, with generator offers denoted by differently shaded bands. Generator offers are collected by

![Figure 1: Electricity price is calculated based on generator supply offers and consumer demand](image-url)

Figure 1: Electricity price is calculated based on generator supply offers and consumer demand
AEMO and stacked in ascending price order. At the time of electricity consumption, generators are scheduled in order of their offer price until the total demand is met. The highest-priced offer required to fulfill the demand is the amount paid to all generators that are scheduled into electricity production. This means that in Figure X at 4:15, generators one, two three and four all receive $37 per MW generated. At this time, generators one, two and three are fully utilised, and generator four is only partially utilised.

1.2.2 Market Power

Economists use a measurement of “market power” to determine how much control a particular entity has on the market it operates within. In some situations, too much market power can lead to catastrophic outcomes, as was experienced in California in 2000 [1]. According to researchers, generators in California in June 2000 abused their market power in an attempt to make more money. After several months, poor market design lead to California’s largest utility, Pacific Gas & Electric (PG&E) declaring bankruptcy in March 2001. In early 2001, prices skyrocketed reaching more than 10 times what they were in 2000.

Preventing this kind market power is important for two reasons. First, if taken to extremes, methods of market control such as flooding the grid or holding back supply could lead to long term damage to power infrastructure. Secondly, as power generation is essentially impossible to completely privatise, governmental oversight is important. The California market breakdown in the early 2000’s lead to California’s government stepping in when PG&E went bankrupt. Although this move was intended to introduce stability to the system, the state government was merely seen as having a larger bankroll than PG&E, and prices rose at an accelerated pace.

This work proposes a software system, Franklin, that uses a multi-agent approach to simulating aspects of Australias National Electricity Market (NEM). The purpose of this simulator is to investigate how fault-tolerant the Australias wholesale electricity market is to natural events such as bushfire and political events (e.g. the introduction of new taxes), to determine the effectiveness of NEMs infrastructure is at preventing an entity from acquiring too much market power, and to promote further research in the field of agent-based computational economics (ACE).

1.3 Scope of Research

This project is intended to produce a reusable multi-agent framework that can be utilised in the study of the Australian Electricity Market.

1.4 Significance

The project’s purpose is to create a software system to simulate the Australia’s National Electricity Market (NEM) in order to evaluate the potential for members of the NEM to
exercise market control by strategic bidding. It will allow researchers to use investigate
the interactions between participants in the market, as well as the effect of various
political and natural events.

The system supports extensive configuration, a flexible event modelling system, and
comprehensive data output and has been designed to be extended by future researchers.
Using the loose “duck-typing” features of the Python programming language, Franklin
is able to achieve excellent flexibility in configuration and functionality.

By building Franklin in an extensible way, developers and future researchers will
be able to implement new agents, operators and events to suit their research projects.
Franklin has been built conforming to the style suggestions in Python’s PEP 8 [2],
which will assist developers in understanding the system.

1.5 Tool Requirements

The system developed by this work must be able to meet the requirements listed in
Table 1.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Models market participants as software agents.</td>
</tr>
<tr>
<td>2</td>
<td>Implements the algorithm used by AEMO to balance supply and demand.</td>
</tr>
<tr>
<td>3</td>
<td>Reads information from data files published publicly by the NEM.</td>
</tr>
<tr>
<td>4</td>
<td>Simulations must be self-sustaining.</td>
</tr>
<tr>
<td>5</td>
<td>Simulations should be configurable with different network architecture.</td>
</tr>
<tr>
<td>6</td>
<td>Simulations should allow for events to be triggered during a simulation run.</td>
</tr>
</tbody>
</table>

Table 1: Franklin requirement listing

2 Proposal

2.1 Software Architecture

2.1.1 Introduction

Franklin was developed in the Python programming language. There are several jus-
tifications for the decision to use this Python: it is considered to be a flexible tool
that supports rapid development and prototyping and is well-regarded as an effective
“glue” language; it has a clear, concise syntax, with indentation enforced by the lan-
guage, useful for improving code readability and preventing coding style inconsistencies
(e.g. the placement of curly braces); and it provides a great deal of functional- and
meta-programming features that are useful in research contexts.
2.1.2 Code Style

To be consistent with Python’s PEP 8 [2], classes in Franklin are named using PascalCase, with methods and functions named in all lower case, and words separated by underscores.

2.1.3 Software Modules

This section lists and describes the various modules that comprise the Franklin system.

Simulation - simulation.py
Contains the classes involved with actually executing the simulation. Typically, classes in this module do not need to be modified by a user, unless they want to change the actual simulation rules.

Agents - agents.py
The Agents module contains code for all agent classes in Franklin. This includes all Generators, Consumers and Operators that take part in a simulation. This file should be modified to add new agents to a simulation or to alter the behaviour of an agent.

Data Monitors - data_monitors.py
This module defines classes that enable one to monitor the output data produced by a simulation run. At this time, the only implemented logger is the CSVFileMonitor, which outputs simulation data in CSV format. Additional data monitor classes are suggested if a future project requires additional data or data in a different format.

Logger - logger.py
This module defines classes that can be used within a simulation to log events as they occur. Currently, the only implemented logger class is BasicFileLogger, which outputs to a flat file.
Note: This module is intended to be merged with the data_monitors module in the future, since it performs a similar function.

Messaging - messaging.py
This module contains the various Message types that are passed between agents via the MessageDispatcher. By adding new kinds of messages, developers can implement new communication protocols between various agents.

Data Providers - data_providers.py
This module contains classes for parsing and loading data into a simulation e.g. PUBLIC_YESTBID data files published publicly by the NEM.

Configuration Utilities - configuration_utilities.py
This module defines the syntax for configuration dictionaries and provides helpful functions for loading, parsing, validating, and running configuration dictionaries.
2.1.4 Typical Usage Flow

Figure 2: Typical use flow for the Franklin system (note: Persistent Storage and Monitor Files represent optional and unimplemented data outputs)

Figure 2 depicts a typical application usage flow. A user prepares a configuration file and passes it to Franklin on the command line. The file is used to configure and initialise the simulation which outputs data to multiple sources while running. At this time, CSV and Monitor files are implemented.

2.1.5 Classes

Figure 3: Simplified class diagram of Franklins architecture (no class members shown)
Figure 3 shows a simplified class diagram of the Franklin architecture. It is missing details for each class, but shows the relationships between classes at a high level. There are classes not shown here that inherit from Message, Event, Generator, Consumer and Operator.

2.1.6 Experiment Configuration

Experiment configuration is performed with a Python script that is specified on the command line and dynamically loaded and validated by Franklin’s configuration engine. The implementation of this feature is discussed further in Section 2.3. Using Python as the format for the configuration file is one of the easiest ways of ensuring that configuration parses and validates correctly. An additional advantage of using this method is that configuration values can be generated programmatically before being passed to the simulation.

2.2 Agents

2.2.1 Architecture

In Franklin’s architecture, agents represent autonomous entities operating within the NEM, such as Generators, Consumers, and Operators. Each “time step” in a simulation, every agent has a `step` function that is called. This function contains the necessary logic for an agent to conduct its observations, decision making, communication with other agents, and other actions throughout the duration of a simulation. Agents must also have a `handle_messages` function, which used to process message received from other agents and respond if necessary, and a `get_initialisation_times` function, which is used for agent initialisation purposes (both detailed later in this report). Because Python employs duck-typing, the architecture has been designed so that agents can be any data type and do not necessarily need to extend the classes defined in Franklin’s architecture - they are provided as a reference. As long as a class has the three aforementioned required functions, it will be usable in a Franklin simulation.

Figure 4 shows a high level representation of the classic agent architecture [3]. This architecture is used by all agents in the Franklin system.

One interesting point to note about agents is that they are, architecturally, not bound to a specific simulation instance – an agent can be instantiated without requiring a reference to a specific simulation instance. In fact, a simulation instance is only ever provided to an agent as an argument to one of its public functions, such as `step` or `get_initialisation_times`, effectively enabling one to pass “facade” simulations to agents if it is necessary (such as for testing or experimental purposes).

2.2.2 Communications

To enable communications between market participants, a message dispatching system was incorporated into the tool. The `MessageDispatcher` system loosely follows the
Mediator pattern, published in [4], where the MessageDispatcher allows for loose coupling between agents interacting in the system. Franklin’s MessageDispatcher also represents an ad-hoc, polled event system, where messages can be retrieved from the MessageDispatcher, rather than having the dispatcher push messages to the agents.

This system consists of a MessageDispatcher object, which acts as a centralised entity for message storage and retrieval, and numerous different message types (e.g. demand forecasts, dispatch notifications, generator bids, pre-dispatch schedules, etc.). A MessageDispatcher stores an inbox of messages per agent, and a simulation has a single MessageDispatcher object instance. Each time step an agent can send messages to another agent’s inbox via the MessageDispatcher, and retrieve messages from its own inbox for processing. This process is shown in Figure 5.

From a multi-agent perspective, one of the interesting facets of the Franklin’s messaging system is the manner in which it enables multiple back-and-forth communications between agents in a single time step. Since a simulation runs at a granularity of “one minute of real-time” per time step, it was deemed potentially limiting to only offer agents the opportunity to send and receive messages once in an entire minute. Although it may seem an improbable scenario, because the NEM operates in real-time it is possible that two or more agents will need to communicate many times within the same simulation step. Hence, a “communications loop” was implemented to allow agents to perform several message exchanges within a given time step, in an attempt to model the real-time aspect of the market as realistically as possible without resorting to increasing the simulation granularity to seconds or milliseconds, which would be more computationally intensive. An example of this back-and-forth message exchange would be if two generators were attempting to collude to achieve a better profit. They would need to use multiple messages to coordinate their bids, possibly negotiating between themselves before making their bids to AEMO. To facilitate this, agents need to be able to have an conversation of arbitrary length within one simulation time step.
Essentially, a loop is performed wherein each cycle of the loop, an agent processes all messages in its inbox (which effectively clears the inbox), and sends messages to other agents if necessary; once all inboxes are empty (i.e. all agents have no more messages to process), the loop exits.

2.2.3 Initialisation

Because the NEM is considered perpetually in operation, Franklin simulations need to “initialise” the market in some manner. Hence, before every simulation, each agent is queried for the dates and times prior to the simulation start date that the market needs to be simulated in order for the agent to initialise itself. The tool then simulates the market for each date returned by an agent. This allows agents to perform actions and setup sequences before the simulation start date that may be necessary.

A common example is generator bidding for the first day of the simulation. The National Electricity Law and Rules [5] dictate that generators can only submit offers before 12:30pm of the previous day that the offer is intended for. Without initialisation, there would be no way for generators to do this prior to the simulation start date, meaning the first day of every simulation would be non-operational, since no generator offers would be present. Clearly, this is at odds with the notion of a market that is always considered in operation. Hence, by asking each agent what dates and times it requires for initialisation, and then simulating those individual times, generators can
submit offers before 12:30pm of the day before the simulation start date if they need to.

2.3 Customisable Simulation Configurations

To enable ease of experimentation and testing, Franklin provides support for customisation of numerous simulation parameters. This includes simulation-specific settings such as market “start” date, end date, regions, consumers, generators, as well more general settings (that may be re-used over multiple simulation runs), such as the logging object to handle simulation output. Simulation configurations are defined programmatically using Python dictionaries, each key-value pair corresponding to a specific simulation setting.

Configuration dictionaries must adhere to a syntax that defines the configurable simulation parameters and their criteria for validity. A configuration validation system handles the rejection of configuration entries that fail to meet their validity criteria, as well as the truncation or conversion of valid values to a more appropriate, desirable format if such is required. As depicted in Figure 6, three aspects comprise validation of configuration parameters: pre-validation, post-processing, and post-validation.

![Figure 6: The process of configuration validation prior to simulation execution](image)

The configuration syntax defines a pre-validator function for each configuration parameter, and this function is called by the configuration validation system to initially determine whether a value is acceptable on its own (i.e. with no comparisons to other configuration entries). This may include verification of whether a value has the required attributes, is a certain type of data structure, is within a certain range, and so on.

If a configuration entry passes pre-validation, a post-processor function defined for each configuration parameter is called to modify and manipulate their values in some manner (e.g. convert a list of specified events into a set to remove duplicates). Generally, the primary reason for post-processing is not validation, but to safeguard a simulation run against user error that may not necessarily be detectable in the validation process.

Lastly, once post-processing is complete, a post-validator function is called for each configuration parameter, which is typically used to cross-validate configuration parameters (e.g. verify that the simulation start date is before the end date).

The configuration syntax also provides default values for parameters in configuration dictionaries in the event that they are not specified by the user, and a deprecated flag.
to indicate when configuration parameters are obsolete or only provided for legacy purposes.

It should be noted that the configuration system is an entirely separate module from simulations. That is, a simulation can be instantiated and executed even by completely bypassing the configuration validation system. Although this is not advised, it provides flexibility for researchers who may integrate Franklin with other tools that implement their own means of configuration validation.

2.4 Events

The NEM is not deterministic and the entities operating within the market can be affected by numerous internal and external factors (e.g. physical, economical, environmental, etc.). Hence, in order to effectively model sudden changes in the market, Franklin provides support for triggering events at particular times throughout simulation runs. Events are user-specified via configuration dictionaries. Like agents, the event model is relatively fluid with its requirements for how events must be defined. An event can be any data type, provided it has the following two attributes: a time\_delta public member that indicates the date/time from the simulation start date that the event should occur, and a process\_event function that can be called to activate the event at that date. The SimulationEvent class included in Franklin is provided as a reference.

A simulation processes events by using an event stack approach, illustrated in Figure 7. Events are sorted by date/time (earliest first), and the event at the top of the stack is popped when the current time of the simulation is equal to the start date of the simulation plus the event’s time\_delta value. When an event is popped from the stack, it’s process\_event function is called, which operates on the simulation instance.

![Event Stack Diagram](image_url)

Figure 7: The event at the top of the stack is only popped when it is time to activate it.
that is passed to the function as an argument. The event is subsequently discarded and the entire process is repeated for the next event (i.e. the new “top of the stack”). The advantage of using a stack-based approach is primarily performance-based – events occurring at later dates need not be considered every time step; in most cases, only the event at the top of the stack needs be checked.

2.5 Data Monitors

Data collection is an important aspect of any research-based software tool. Franklin supports the notion of data monitors – that is, objects that track and log the data output of a simulation. Whether the data is trading interval spot prices, energy bids per region, or total demand at a given dispatch interval, a data monitor can be used to facilitate its collection. Data monitors are specifiable via configuration dictionaries. Currently, the tool only permits post-simulation data collection via calling a data monitor’s log_run function, however the basic data monitoring framework could easily be extended to operate at different levels (e.g. every time step, every dispatch interval, etc.) if such enhancements were required.

2.6 Data Providers

Franklin has built-in support for loading data published by the NEM. Currently, there are two classes that read NEM data from file: CSVPublicYestBidDataProvider, which parses bid data from PUBLIC_YESTBID data files, and CSVPublicPricesDataProvider, which parses demand and pricing data from PUBLIC_PRICES files. These classes have been implemented to provide an example of how NEM data can be parsed and converted into a format that is usable in a Franklin simulation.

3 Discussion

3.1 Existing Systems

The AMES system developed by Koesrindartoto and Tesfatsion is the primary inspiration for developing Franklin. In terms of completeness, the AMES system models more of the electricity market and some of the physical aspects of the electrical grid, implementing a Locational Marginal Pricing scheme for the market. It is based on the New England power market developed in the United States and operated by the New England Independent System Operator [6].

3.2 Limitations

The Franklin system implements a basic level of simulation of the energy market. Interactions between agents are currently only used for communicating generator offers
and consumer demand forecasts, but this could easily be extended in future research projects.

The electrical power markets in Australia and around the world are intricate and complex. For Franklin, only the relevant aspects have been extracted and modelled, for the purposes of running a self-sustaining simulation of the market. Factors such as generator “ramp up” time, equipment maintenance, frequency shifts and economic flows are outside the scope of this basic simulation, and were not implemented in this instance. Future work that requires additional information or data would need to implement these features, after determining what the relevant variables that require modelling are.

3.3 Future Work

3.3.1 Projects

In future, Franklin could be extended to perform a more accurate modelling of the electrical power market in some aspect. Determining the aspects to be modelled will depend on the specific project that is being undertaken. As an example, a study investigating the effect of transmission interruption due to bushfire would need to model at least some of the physical electrical grid structure, to determine the critical edges in the graph representation of the electrical grid. A project that investigates how ancillary services can be used to acquire market power would need to model the provision of these services and potentially the effects of the services on the network.

3.3.2 Framework

Franklin’s feature set as a framework is nearly complete with respect to the original goals of the project. It has an extensive set of logging and configuration options, which are also extensible to allow customisation for various research projects. Franklin can also be integrated with other Python-based tools, and it provides researchers with the ability to dynamically configure simulations at run-time (as opposed to always having permanently static, scripted configurations), as well as retain memory handles on any objects passed into a simulation even after it has concluded execution.

4 Conclusion

Franklin was designed and built to investigate the effects of various events and changes in the Australia’s National Electricity Market (NEM). Franklin simulates the interactions between entities operating within the market using multi-agent approach. This system produces data on the allocation of generating units in the electrical system based on the sale price offered by each unit, and could be extended in future works to more comprehensively model the wholesale electricity market.
5 References


