Enhancing Agent Intelligence through Evolving Reservoir Networks for Predictions in Power Stock Markets

Kyriakos C. Chatzidimitriou[1,2], Antonios C. Chrysopoulos[1], Andreas L. Symeonidis[1,2], and Pericles A. Mitkas[1,2]

[1] Electrical and Computer Engineering Department
Aristotle University of Thessaloniki
GR 54124, Thessaloniki, Greece
[2] Informatics and Telematics Institute
Centre for Research and Technology Hellas
GR 57001, Thermi, Thessaloniki, Greece
{kyrcha,achryso}@issel.ee.auth.gr,
{asymeon,mitkas}@eng.auth.gr
http://issel.ee.auth.gr

Abstract. In recent years, *Time Series Prediction* and clustering have been employed in hyperactive and evolving environments –where temporal data play an important role– as a result of the need for reliable methods to estimate and predict the pattern or behavior of events and systems. *Power Stock Markets* are such highly dynamic and competitive auction environments, additionally perplexed by constrained power laws in the various stages, from production to transmission and consumption. As with all real-time auctioning environments, the limited time available for decision making provides an ideal testbed for autonomous agents to develop bidding strategies that exploit time series prediction. Within the context of this paper, we present *Cassandra*, a dynamic platform that fosters the development of Data-Mining enhanced Multi-agent systems. Special attention was given on the efficiency and reusability of *Cassandra*, which provides Plug-n-Play capabilities, so that users may adapt their solution to the problem at hand. *Cassandra*'s functionality is demonstrated through a pilot case, where autonomously adaptive *Recurrent Neural Networks* in the form of *Echo State Networks* are encapsulated into *Cassandra* agents, in order to generate power load and settlement price prediction models in typical *Day-ahead Power Markets*. The system has been tested in a real-world scenario, that of the Greek Energy Stock Market.

Keywords: Data Mining, Power Stock Markets, Reservoir Computing, Multi-Agent System, Neuroevolution.

1 Introduction

Agent Technology (AT) has been successfully employed in various aspects of real-world trading and electronic markets [12]. In highly dynamic, uncertain and

L. Cao et al. (Eds.): ADMI 2011, LNAI 7103, pp. 228–247, 2012.

multi-player markets, decisions have to be made continuously, within limited time, while data are generated at extreme rates. Thus, agents are considered a suitable candidate for building efficient trading mechanisms, where their intelligence is based on algorithms that are able to adapt, requiring no or limited user input.

Based on authors' prior work [24], we argue that software agents can be employed with Data Mining (DM) capabilities [6], embed useful nuggets of knowledge, and gain a predictive advantage over their competitors. Performing DM on the (residing or streaming) data pool, one may model the problem at hand, validate a hypothesis, or even predict trends and behaviors. Once the pattern is established, one can then interpret and integrate it with other data (i.e., use it in the theory of the investigated phenomenon, e.g., seasonal commodity prices). Regardless of the depth of the understanding and the validity of the interpretation (theory) of the phenomenon, one can extrapolate the identified pattern to predict future events.

To this end, we have fused AT and DM and developed *Cassandra*, an Agent-Based Framework that fully supports Data Mining capabilities for Prediction and Rapid Problem Solving. *Cassandra* adopts a modular, loosely-coupled, 4-layer Service Oriented Architecture (SOA) [4], with *Plug-n-Play capabilities*. Interconnections between layers, as well as the communication interfaces with users or other applications / Web Services, are properly defined. *Cassandra* provides a step-by-step, easy-to-use guide, as well as a respective Web Services Description Language (WSDL[1]) specification and data structure, in order to encapsulate prediction models (PMML[2]). Through *Cassandra*, the user may select the appropriate dataset(s), preprocess it, and select a Data Mining API to build his/her models (WEKA and R are fully supported, other APIs are partially supported also). Results are displayed and stored, while action can be taken (in an autonomous or semi-autonomous manner), when deemed appropriate.

We prove the feasibility of our approach through a demonstrator case for the Greek Power Stock Market, which is a dynamic, partially observable environment, giving room for applying a wide variety of strategic approaches. In order to capture the temporal and non-linear dynamics of the Power Stock Market signals, we employ *Echo State Networks* (ESNs), a state-of-the-art neural network function approximator. In order to promote the autonomy of the multi-agent system deployed, networks are self-adaptive through neuroevolution [22]. Short-term predictions are made for load and price time-series and are tested against standard regression techniques, previously employed in the *Cassandra* system. Results with respect to load forecasting were excellent, while for price forecasting, which is a much more complex time-series, promising.

The rest of the paper is organized as follows: Section 2 provides the background theory with respect to Echo State Networks and Power Market auction environments. Section 3 discusses related work on platforms enhanced with DM capabilities, as well as AT and DM approaches for solving problems related to

[1] http://www.w3.org/TR/wsdl20/
[2] http://www.dmg.org/v4-0/GeneralStructure.html

Power Markets. Section 4 presents the *Cassandra* architecture and functionality, while Section 5 discusses the problem domain, the proposed solution, the experiments conducted and the results derived. Finally, Section 6 summarizes work and concludes the paper.

2 Background Theory

2.1 Echo State Networks

The idea behind *reservoir computing* (RC) and in particular Echo State Networks (ESNs) [13] is that a random *recurrent neural network* (RNN), created under certain algebraic constraints, could be driven by an input signal to create a rich set of dynamics in its reservoir of neurons, forming non-linear response signals. These signals, along with the input signals, could be combined to form the so-called *read-out function*, a linear combination of features, $y = \mathbf{w}^T \cdot \phi(\mathbf{x})$, which constitutes the prediction of the desired output signal, given that the weights, w, are trained accordingly.

The basic structure of an ESN is depicted in Figure 1. The reservoir consists of a layer of K input units, connected to N reservoir units through an $N \times K$ weighted connection matrix W^{in}. The connection matrix of the reservoir, W, is an $N \times N$ matrix. Optionally, an $N \times L$ backprojection matrix W^{back} could be employed, where L is the number of output units, connecting the outputs back to the reservoir neurons. The weights from input units (linear features) and reservoir units (non-linear features) to the output are collected into an $L \times (K + N)$ matrix, W^{out}. For this, the reservoir units use $f(x) = tanh(x)$ as an activation function, while the output units use either $g(x) = tanh(x)$ or the identity function, $g(x) = x$.

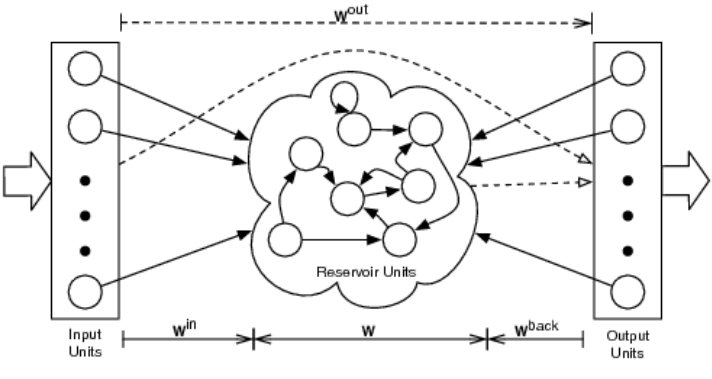


Fig. 1. A basic form of an ESN. Solid arrows represent fixed weights and dashed arrows adaptable weights.

One may refer to [13,14] for best practices for generating ESNs, in the sense of procedures for generating the random connection matrices W^{in}, W and W^{back}. These could be briefly summarized in the following: (i) W should be sparse, (ii) the mean value of weights should be around zero, (iii) N should be large enough to introduce more features for better prediction performance, (iv) the spectral radius, ρ, of W should be less than 1 to practically (and not theoretically) ensure that the network will be able to function as an ESN. Finally, a weak uniform white noise term can be added to the features for stability reasons.

In current work, we consider discrete time models and ESNs without backprojection connections. As a first step, we scale and shift the input signal, $\mathbf{u} \in \mathbb{R}^K$, depending on whether we want the network to work in the linear on the nonlinear part of the sigmoid function. The reservoir feature vector, $\mathbf{x} \in \mathbb{R}^N$, is given by Equation 1:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(t+1) + \mathbf{W}\mathbf{x}(t) + \mathbf{v}(t+1)) \tag{1}$$

where $\mathbf{f}$ is the element-wise application of the reservoir activation function and $\mathbf{v}$ is a uniform white noise vector. The output, $\mathbf{y} \in \mathbb{R}^L$, is then given by Equation 2:

$$\mathbf{y}(t+1) = \mathbf{g}(\mathbf{W}^{out}[\mathbf{u}(t+1)|\mathbf{x}(t+1)]) \tag{2}$$

with $\mathbf{g}$, the element-wise application of the output activation function.

2.2 NeuroEvolution of Augmented Topologies

NeuroEvolution of Augmented Topologies (NEAT) [21] is a topology and weight evolution algorithm of artificial neural networks, founded upon four principles that have established it as a reference algorithm in the area of NeuroEvolution. First of all, the network, i.e. the phenotype, is encoded as a linear genome (genotype), making it memory efficient with respect to algorithms that work with full weight connection matrices. Second, using the concept of *historical markings*, newly created connections are annotated with innovation numbers. During crossover, NEAT aligns parent genomes by matching the innovation numbers and performs crossover on these matching genes (connections). The third principle is to protect innovation through *speciation*, by clustering organisms into species in order for them to have time to optimize by competing only in their own niche. Last but not least, NEAT initiates with minimal networks, i.e. networks with no hidden units, in order to: (a) initially start with a minimal search space and, (b) justify every complexification made in terms of fitness. NEAT complexifies networks through the application of structural mutations, by adding nodes and connections, and further adapts the networks through weight mutation by perturbing or restarting weight values. The above successful ideas could be used in other NE settings in the form of a meta-search evolutionary procedure. Within the context of our work, we adopt the NEAT model in order to achieve an efficient search in the space of ESNs.

2.3 NeuroEvolution of Augmented Reservoirs

NeuroEvolution of Augmented Reservoirs (NEAR) utilizes NEAT as a meta-search algorithm and adapts its four principles to the ESN model of neural networks. The structure of the evolutionary search algorithm is the same like in NEAT, with adaptations made mainly with respect to gene representation, crossover with historical markings and clustering, thus including some additional evolutionary operators related to ESNs. A major differentiation from NEAT is that both evolution and learning are used in order to adapt networks to the problem at hand. A complete description of NEAR can be found in [7].

2.4 Power Market Auctions

The deregulation of the Power Market has given room for the development of open markets, where participants are able to choose between different energy products in different periods of time and may negotiate on their "product portfolio". These portfolios pertain to three different market regimes:

- *The Long-term Market*, where participants come to direct agreements in form of long-term contracts.
- *The Day-ahead Market*, where buyers place their bids in 24 hourly auctions, in order to establish a contract for the next day.
- *The Real-time Market*, where buyers place their bids in order to establish a contract for the next hour.

In Power Market Auction environments, two are the most important entities:

1. *The Market participants* (or Players)
2. *The Independent System Administrator* (ISA)

A *Player* is defined as a financial entity that accesses the Power Market [25]. In general, this entity may represent a group of Production Units and/or a group of Consumers. Players participating in the Power Market as *Producers*, submit their power supply offers in pre-specified time intervals. Each offer contains the amount of supplying Power, as well as the minimum price one is willing to accept. On the other hand, Players participating in the Power Market as *Consumers* submit their power demands within the same time intervals, along with the maximum price they are willing to pay for it.

The *ISA* is the administrator of the Power System, and also the Administrator of the Power Stock Market (more entities may exist, but they are merged into one for simplicity reasons). Among others, ISA is responsible for the *Settlement Price* of the Power Market, taking into consideration the limitations of the power system. ISA collects bids for each auction and calculates two curves: the *aggregate Supply Curve* (ascending) and the *aggregate Demand Curve* (descending). In the simplified case, where no transmission limitations exist, Settlement of the Market is achieved at the intersection of the two curves (Figure 2). The intersection point specifies the Settlement Price of the Market (SPM), as well as the load to be provided/offered.

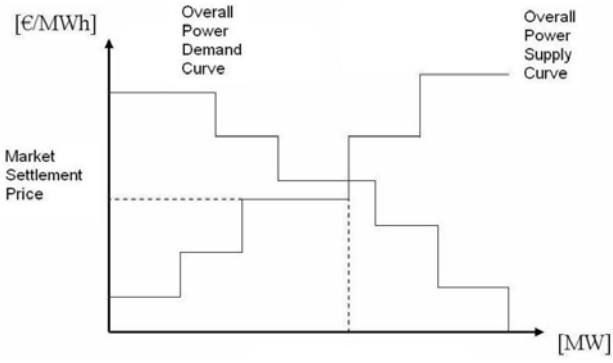


Fig. 2. The aggregate power supply and demand curves

Though demand is mostly covered by Long-term and Day-ahead market transactions, the fact that power cannot be stored for long periods of time dictates the development of a mechanism that can efficiently balance supply and demand of power. Such a balance between production and consumption is ensured through the utilization of a Real-Time Market model. This model bears the highest risk for participants, since the malfunction of an electric node or the shutting down of a power line can bring exaggerated rising or falling of loads and prices. Nevertheless, higher risks also imply profit maximization potential for players willing to participate in this market.

3 Related Work

3.1 DM-Enhanced Software Platforms

There have been several attempts to provide semi-autonomous platforms that fuse Agent Technologies, Data Mining and later on, Web Services.

In the turn of the new millennium, Web Service (WS) technology came to the foreground, resulting to a bloom of platforms based on this new technological achievement. One of the first noticeable implementations is SWORD [18], a set of tools for the composition of a class of web services including "information-providing" services. Nevertheless, the data integration capabilities of the platform did not support standards like WSDL and SOAP, weakening openness and interoperability characteristics of the system.

Following, an agent-based architecture for autonomic web service selection was proposed [15]. This approach selected the appropriate web service from the service directory, employing an aggregated system based on other users' feedback, trusted sites' opinion and previous results of the same web service usage. Nevertheless, the platform was lacking composition capabilities, still leaving no room for extensibility.

Working the other way around, some research efforts exist, attempting to build Agent-based Web Services. The Personal Agent Framework for Web Services and Commercial Systems [19] should also be referred to at this line of

work. It is a platform-independent, hybrid artificial intelligent agent framework. Its purpose is to assist a user with processing and retrieving of emails, files, data (from databases), and recreational requests such as searching for an appropriate restaurant or making show reservations, all through a single user interface (portal). The module-based approach is not only appropriate from a software engineering point of view, but it also allows proven artificial intelligence technologies to be incorporated, for example, neural network and automatic speech recognition models.

Finally, special note must be made to the work of Diosteanu and Coltfas [10] for implementing an Agent Based Knowledge Management Solution using Ontology, Semantic Web Services and GIS. Within the context of their work, they developed an agent based knowledge management application framework using a specific type of ontology that is able to facilitate semantic web service search and automatic composition. This framework was later on used to develop complex solutions for location based services, supply chain management, etc, though its main orientation was economic applications.

3.2 Power Market Analysis through AT and DM

Various approaches have been employed for analyzing the behavior of Power Markets, some of which employ AT and DM primitives.

In the early 2000s, during the bloom of MAS utilization, the Electric Power Research Institute (EPRI) developed SEPIA (Simulator for Electrical Power Industry Agents), a multi-agent platform capable of running a plethora of computing experiments for many different market scenarios [2,1]. The Argonne National Laboratory, on the other hand, developed EMCAS (Electricity Market Complex Adaptive System) [8], an efficient implementation for handling the Electric Energy Market. Through EMCAS, one may study the complex interactions between the physical entities of the market, in order to analyze the participants and their strategies. Players' learning is based on genetic algorithms, while EMCAS supports stock market transactions, as well as bipartite contracts. Finally, Petrov and Sheble [17] introduced genetic programming in their simulation and tried to model the bipartite Auctions for the Electric Energy Market, by the use of agents. One of the players incorporates knowledge represented as a Decision-Making Tree, which is developed by genetic programming. The rest of the agent-players incorporate ruled-defined behaviors.

Special attention should be drawn to work by Melzian [16], who developed EMSIM (Energy Market SIMulator), in an effort to derive a deeper understanding of the bidding behavior at the EEX (European Energy Exchange), the impact of changes in market design and individual strategies of the participants. Additionally, work by Bremer et al. [5] should also be noted. They built an agent-based environment for modeling and simulating adaptive household consumers responding to dynamic electricity pricing. Focus was given on the analysis of household load shifting potential under different tariffs and negotiation strategies.

Although efficient, the framework approaches mentioned in the state-of-the-art chapter are either too focused on a specific application domain or a specific learning mechanism. From a software engineering perspective, one may identify that they cannot adapt to new specifications, they are not expandable and modular, and what is most important, cannot reuse the existing infrastructures. From a learning/adaptation perspective, one may identify that they cannot dictate change in the course of action (unless the user directly asks them to). Instead of focusing on the exploitation of Data Mining extracted knowledge in order to augment agent intelligence and autonomy, existing approaches merely focus on solving a very specific problem or selecting the right web service for the given application.

This is the main reason why *Cassandra* is built as a general-purpose prediction tool. Different prediction types and different DM algorithms are supported, while *Cassandra* can manipulate various data types and various datasets. We demonstrate the power of *Cassandra* by focusing on the newly added algorithms, ESNs and NEAR. With respect to power load forecasting, ESNs have been studied in [20,3], where data were drawn from the "World-wide competition within the EUNITE network" [3]. Neither of the approaches optimize topology, reservoir properties and weights at the same time, in order to adapt the function approximator with minimum human intervention. Additionally, besides load forecasting, we deal with settlement price forecasting as well, which is a much more demanding problem.

4 The Cassandra MAS

4.1 Cassandra Scope of Use

Cassandra follows a hybrid Agent Oriented Software Engineering approach, integrating AT, DM and WS technologies, in order to provide a powerful schema for Prediction and Rapid Problem Solving. Combining the AT approach with WS leads to a uniform representation of services and offer a greater degree of interoperability when using heterogeneous agents and service providers. On the other hand, DM is employed in order to generate knowledge models from data, which can then be dynamically embedded into agents. As new data accumulate, the process can be repeated and the decision structures can be updated, effectively retraining the agents and improving system efficiency.

Looking at the bigger picture, the fusion of these three leading technologies can lead to a wide spectrum of autonomous data mining-enhanced tools covering most business and personal needs. On this basis, *Cassandra* can reach its full potential on environments with sufficient amount of data available (real time data, historical data, correlated data storages etc.) utilized for DM purposes. Based on the processed data, autonomous models are developed, making *Cassandra* an invaluable tool in use cases where such Business Intelligence implementations are needed.

[3] http://neuron.tuke.sk/competition/index.php

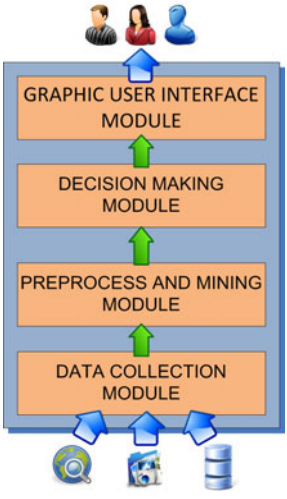


Fig. 3. Cassandra 4-Layer Architecture

4.2 Architecture

Cassandra follows the IRF Architecture Model (Intelligent Recommendation Framework) [23], which defines a 4-layer functional model for the agent system. IRF is usually employed in enterprises for the optimization of the administration mechanism, since it can automate and integrate all the data producing or data demanding facets of a company. Taking a closer look of the Power Market through the IRF prism, one may identify several tasks that have to be tackled:

- Collection of the historical data from previous auctions and processing of the data.
- Application of the suitable DM algorithms in order to build the necessary forecasting models for the values in question.
- Integration of generated models in the Business Intelligence of the System and evaluation of the results.
- Continuous monitoring of the power stock market in order to validate the efficiency of the platform.

As expected, *Cassandra* employs a modular architecture (Figure 3), where each module is responsible for one of the aforementioned tasks. The platform also provides a wrapper around all modules and ensures communication with the system users. The modules comprising *Cassandra* are:

1. Data Collection Module (DCM): Collecting historical data, either from files provided by the user, or directly via an API or the web.
2. Data Processing and Mining Module (DPMM): Processing datasets, preparing training sets and applying DM algorithms.

3. Decision Making Module (DMM): Aggregating all raw data and derived knowledge models in order to make the optimal decision in any given occasion.
4. Graphic User Interface Module (GUIM): Interacting with the System Users. It must be user-friendly, and easily comprehensive.

The most important feature of *Cassandra*, though, is its Plug-N-Play capability; it has the ability to replace any of the four basic modules easily, thus allowing for full expansion of the tool.

4.3 Cassandra User Roles

Cassandra identifies three types of user roles, each accommodated through a respective system view. Table 1 summarizes the functionality provided to each role, enabled upon user authentication.

Table 1. Cassandra User Roles and their respective functionality

User Type	Views	Privileges	Functionality
Operator	- Managing Only	None	- Account Checking - Interface Monitoring - WWW / RSS Browsing - Statistic / Chart Viewing
System Analyst	- Monitoring - Manager	Managing only	*All of the above plus:* - Manual Decision Making - Logger Handling
System Architect	- Monitoring - Manager - Analyst	All	*All of the above plus:* - Agent Overview - Model Retraining - Model Configuration - Model Statistics - Cost Evaluator - Data File Import

- **System Architect.** The *System Architect (SAr)* is an expert user. SAr is the developer of the system, so he/she has absolute knowledge over it. SAr is responsible for the smooth operation (unimpeded operation of the system, satisfaction of the software requirements of the users), as well as its efficiency. SAr also provides domain knowledge, while he/she is responsible for generating the DM models, in order to decide on the best-performing one(s). Finally, SAr checks the decision routines, in order to trace any errors that may occur.
- **System Analyst.** The *System Analyst (SAn)* is a domain expert, not necessarily on the involved technologies, but on the application at hand. SAn cannot interfere directly with the system, but using his/her experience SAn

can easily assess *Cassandra* actions and decide whether the system operates efficiently enough or not. SAn has the authority to manually override *Cassandra* decisions, but he/she cannot change the models used by the system for prediction (that is under the Architect's jurisdiction).

- **Operator.** The *Operator* is the simple user of the system. Operator only overviews and logs actions and decisions made by *Cassandra*. In case the Operator notices something "out of the ordinary" (actions that do not have the expected results), he/she notifies the SAn. SAn must then double check Operators observations and, in case of error, notify the SAr, who will react accordingly in order to optimize system operation.

In order to ensure robustness and reusability, *Cassandra* employs a hybrid architecture comprising a set of auxiliary modules ensuring untroubled integration and easy module replacement, as well as a MAS configuration for building intelligent applications, incorporating domain intelligence and performing DM. The *Cassandra* abstract level Z-Notation (Figure 4) depicts the entities involved and the main *Cassandra* goals.

$$
\begin{array}{|l}
\hline
\textit{Cassandra} \\
\hline
\textit{MultiAgentSystem} \\
\textit{WhitePages} \\
\textit{YellowPages} \\
\textit{AuthenticationConsole} \\
\textit{Logger} \\
\textit{members} : \mathbb{P}\ \textit{CassandraAgent} \\
\textit{Name} : \textit{CassandraAgent} \rightarrowtail \textit{String} \\
\textit{Type} : \textit{CassandraAgent} \rightarrowtail \textit{AgentType} \\
\textit{AllocatedStatus} : \textit{CassandraAgent} \rightarrowtail \textit{AgentStatus} \\
\textit{Behave} : \textit{CassandraAgent} \leftrightarrow \textit{CassandraBehaviour} \\
\textit{cassandraGoals} : \mathbb{P}_1\ \textit{CassandraGoal} \\
\hline
\text{dom}\ \textit{Name} \subseteq \textit{members} \\
\text{dom}\ \textit{Behave} \subseteq \textit{members} \\
\text{dom}\ \textit{WhiteEntries} \subseteq \textit{members} \\
\text{dom}\ \textit{YellowEntries} \subseteq \textit{members} \\
\textit{cassandraGoals} = \{\textit{SolveProblem}, \textit{MakeDecisions}\} \\
\hline
\end{array}
$$

Fig. 4. Schema of the Cassandra platform in Z-Notation

4.4 Cassandra Agent Types and Auxiliary Modules

Cassandra comprises four subsystems, each containing a number of agents (members) with different goals and behaviors. Agents are built upon four basic concepts, well-defined in the schema proposed by *d' Inverno and Luck* [9]:

1. Attributes: a set of perceivable features, varying for every type of agent.
2. Actions: a set of discrete events that can change the state of the environment when performed
3. Goals: a set of states of affairs to be achieved in the environment.
4. Motivations: a set of desires or preferences that may lead to the generation and adoption of goals and that affect the outcome of the reasoning or behavioral task intended to satisfy those goals.

The following attributes are common for all *Cassandra* agents:

- *Name*: Distinctive and unique in order to characterize each agent.
- *Type*: There are many different types of agents developed within the context of *Cassandra*. These types are generic and can be adapted or even replaced, given the user needs. Table 2 lists the supported types and their main functionality.
- *Status*: Agents can be *Ready*, waiting to be assigned with a task to perform, *Busy*, i.e. agents already assigned with a task, *Suspended*, which have postponed operation for the time being, and *Disabled* agents, which are unable to operate.

Table 2. Cassandra Agent Types

Agent Type	**Description**
Data Collector	Collecting Data From Internet, Databases or Files.
Data Transmitter	Converting files into different extensions and transmitting data to the appropriate agent.
Data Cleaner Data Transformer Data Integrator Data Reducer Data Discretizer	Filtering datasets usign various filtering methods when needed.
Curves Agent	Simulating prediction curves including noise and outliers.
Models Agent	Creating and encapsulating prediction models to agents.
Cost Agent	Estimating prediction costs and efficiency.
Launching Agent	Responsible for the initialization of all the other *Cassandra* Agents.
GUI Agent	Responsible for the Graphic User Interface of *Cassandra*.
Decision Making Agent	Taking decisions after considering every available parameter of the system.

Cassandra implements four auxiliary modules in order to accommodate the basic operations of the system and agents are successfully integrated with them. These objects are the *WhitePages*, *YellowPages*, *Authentication Console* and the *Logger*.

- The *White Pages* register each new member-agent upon instantiation, while also monitor agents' *Status*. *Enter* and *Exit* functionality defined in the *White Pages* interface correspond to the respective actions.

- *Yellow Pages* are operate in a similar to real-life manner, categorizing agents by type. Whenever a user is in search of a certain type of agent, all he/she has to do is query the *Yellow Pages*. A list of the available agents of the type specified will appear.

- The *Authentication Console* provides the gateway for users to connect and defines the privileges for the different platform roles. The lists of logins and roles can either be hard coded or dynamically defined by the System Architect.

- Finally, the *Logger* is a mechanism that provides a detailed overview of the platform's predictions and actions, in order to monitor and evaluate the efficiency of the system. It also provides an override mechanism for manually bypassing the automated decision making process, in case the System Analyst considers that the prediction made by the *Cassandra* models is not correct.

5 Experiments

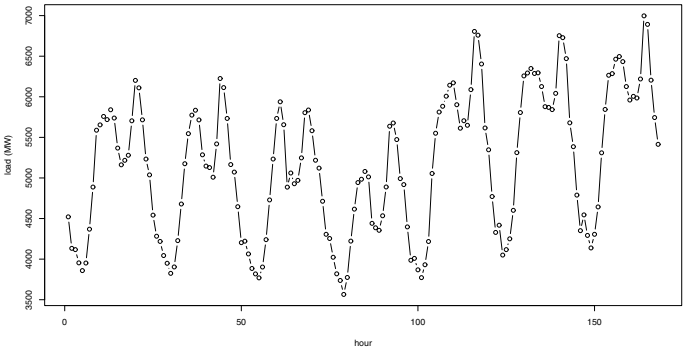
To demonstrate the capabilities of the *Cassandra* platform, we have set up a real-life scenario, based on historical data from the Greek Energy Stock Market (http://www.desmie.gr/). Figure 5 displays the two time series under considerations, power load and settlement prices, for one week period. One may easily observe how irregularities are more intense in Figure 5b than in Figure 5a, indicating a much more difficult task.

Cassandra employs DM algorithms in order build models capable of forecasting the settlement price, as well as the power load for the day-ahead market. The implemented system may even function in a fully autonomous manner, if granted permission, and may proceed with the necessary actions for establishing a contract. The efficiency of the system is highly dependent on the models generated from historical data, as well as a set of the fail-safe rules specified by the Power Market expert (*Cassandra* System Analyst). Through the *Cassandra* interface (Figure 6), DM models produced are rebuilt dynamically, in order to study, evaluate and compare the results and finally choose the one that optimally projects running market trends (*Cassandra* System Architect).

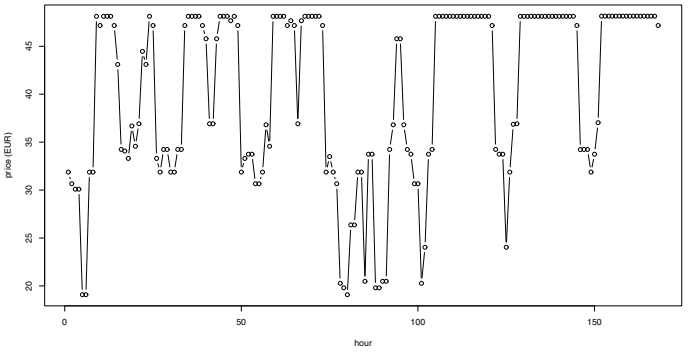
The collected data span from the 1st of March 2003 to the 30th of September 2010. We have devised two tasks, one for each of the two time series. The tasks are to make daily predictions, for a period of one month. This type of task is used to evaluate the *Cassandra* framework as it would have performed in real life, during one month of simulated operation.

Bids by market participants are sent per hour, bundled over a 24-hour period from time $00:00$ to $23:00$ of a single calendar day. Time-series data are available daily, on day d, for that day, after the market clears. They include hourly power load and settlement price values, from time 0h to 23h. Each day the Data Collector agent retrieves the newly available data and stores them to

the repository. The goal is to predict all the 24 hourly values for the next day, $d+1$, knowing only data available up to date d. Figure 7 presents *Cassandra*'s daily cycle.



(a) Load



(b) Price

Fig. 5. Power load and settlement price time series for an indicative one week period

We tested the reservoir computing approaches versus benchmark regression algorithm that exist in the WEKA platform [11]: a) linear regression and b) M5' regression trees. We formulated each task as to build models that would predict the value at hour i given all hour values of the previous day:

$$\hat{x}_i^d = f(x_0^{d-1}, x_1^{d-1}, \ldots, x_{23}^{d-1}), i = 0 \ldots 23$$

where f is the function derived by each algorithm approximating the requested value.

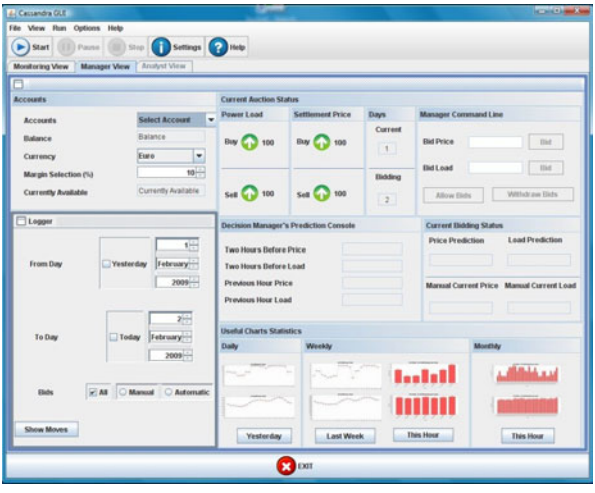


Fig. 6. System Analyst View

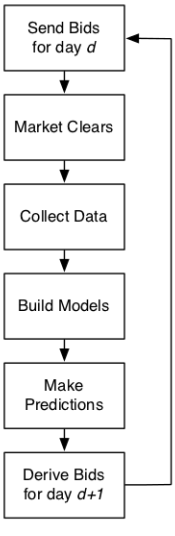


Fig. 7. *Cassandra*'s daily cycle

Performance was measures using the Mean Absolute Percentage Error (MAPE) and the maximal error (MAXIMAL) metrics, which are given by equations:

$$MAPE = 100 \cdot \frac{\sum_{i=1}^{N} \sum_{j=1}^{H} \frac{|L_{Rij} - L_{Pij}|}{L_{Rij}}}{N \cdot H}$$

and

$$MAXIMAL = max(|L_{Rij} - L_{Pij}|)$$

where L_{Rij} is the real value at day i and at hour j and L_{Pij} the predicted value.

For the reservoir computing related algorithms (ESN and NEAR), we have used one year (365 samples) as a washout sequence in order to initialize the networks and minimize initial transient phenomena and two years (730 samples) as training data. MAPE and MAXIMAL errors were calculated for each day. Errors are reported over one month period, that of September 2010 (30 samples). In the NEAR case, fitness was calculated for the month prior to the prediction month (October 2010). The fitness function was defined as $1/MAPE$. A population of 50 networks evolved over 100 generations. The same number of initial reservoir neurons were used as in the ESN case. For the ESN approach each network has 25 inputs, one for each of the 24 hour values of the previous day plus a bias input, and 24 outputs stemming out of the reservoir each one estimating the power load of the next day for a predefined hour. ESN parameters used can be found in Table 5. For the standard supervised learning methods (linear regression and M5') 24 datasets are created and 24 models are derived using them, one for each output. The M5' and linear regression implementations are the ones provided by the WEKA API [11].

Table 3 presents average (AVG) and standard deviations (STD) of the test errors over the period of one month, September 2010, for all the algorithms. The MAPE error is low and indicates a quite precise prediction of the power load time series. The reservoir computing paradigm is superior to the other two algorithms due to its ability to maintain a memory and thus calculate temporal features, important to time series predictions. The NEAR methodology was able to optimize the reservoirs and their properties and as a consequence drive the errors even lower.

Table 4, displays the MAPE and MAXIMAL errors for settlement price forecasting. Again NEAR is providing the *Cassandra* platform with the lower errors, but in general MAPE and MAXIMAL error rates are high. This is due to the highly non-linear and dynamic case of settlement price time series. The market clearing protocol is based on vast volumes of laws that make its prediction a very difficult task. On the other hand, load forecasting is based on the behavior of consumers which is much more predictable.

In Figure 8, we present the forecasts made for both power load and market price on the 24th to the 30th of September 2010 period, using the NEAR methodology. The actual and predicted values in Figure 8a are indicative of the

Table 3. MAPE and MAXIMAL error averages (AVG) and standard deviations (STD) over the 30 days period for all the algorithms evaluated for power load forecasting

Method	**MAPE AVG (%) 30 Days**	**MAXIMAL (MW) 30 Days**
ESN	2.14	316.13
NEAR	**1.91**	**280.14**
Lin. Regression	3.08	436.32
M5'	3.03	570.97
Method	**MAPE STD (%) 30 Days**	**MAXIMAL STD (MW) 30 Days**
ESN	1.20	155.16
NEAR	0.87	114.66
Lin. Regression	1.85	163.57
M5'	1.16	178.49

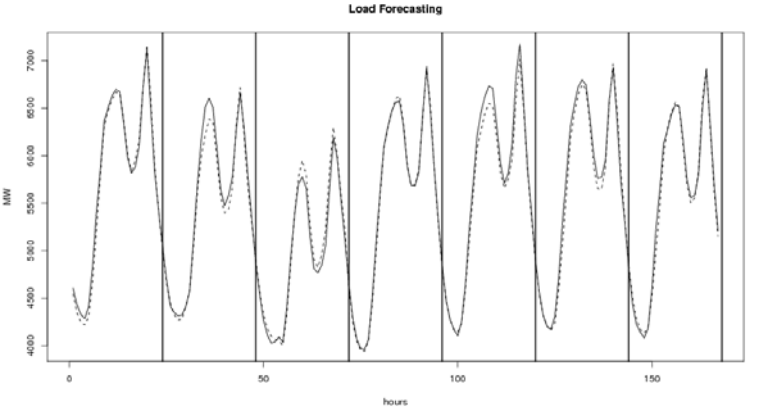
Table 4. MAPE and MAXIMAL error values for ESN and NEAR algorithms with respect to settlement price forecasting

Method	**MAPE AVG (%) 30 Days**	**MAXIMAL STD (Euro) 30 Days**
ESN	18.40	28.00
NEAR	**16.92**	**27.22**
Lin. Regression	17.58	27.56
M5'	17.00	27.54
Method	**MAPE STD (%) 30 Days**	**MAXIMAL STD (Euro) 30 Days**
ESN	6.99	7.82
NEAR	6.27	8.41
Lin. Regression	8.25	8.01
M5'	7.67	7.93

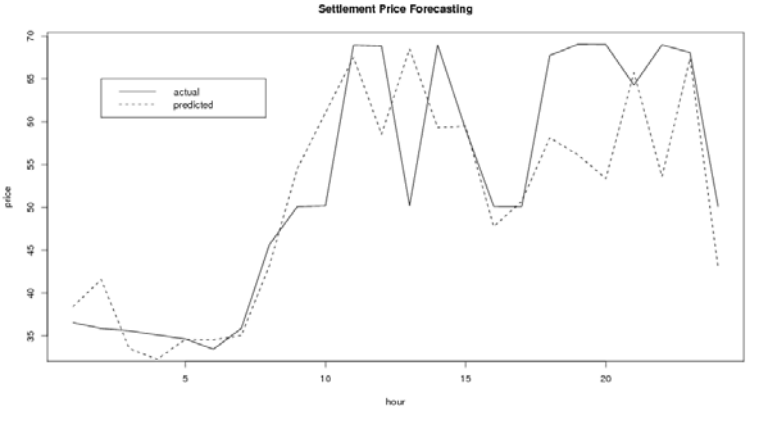
Table 5. ESN parameters initially used (Init.) and ESN parameters derived through NEAR (Found)

ESN Parameter	**Load: Init. (Found)**	**Price: Init. (Found)**
Spectral Radius	0.85 (0.79)	0.85 (0.12)
Density	25% (11%)	25% (9%)
Reservoir Activation Function (f)	$tanh$	$tanh$
Output Activation Function (g)	$identity$	$identity$
Reservoir Size	200 (232)	50 (57)
Noise level	10^{-7}	10^{-5}

prediction abilities of ESNs. In the second case, that of Figure 8b, one may identify that the prediction model can follow highly non-linear time series, nevertheless cannot capture unexpected events. In order to do so, more information on the physical constraints of the power system are needed as well as knowledge on the market clearing protocol.



(a) Weekly load forecasting



(b) Daily price forecasting

Fig. 8. Load forecasting between the 24th and 30th of September 2010 and price forecasting for the 30th of September 2010

6 Conclusions and Future Work

In this paper, we have discussed framework *Cassandra* as a tool for building MAS enhanced with DM capabilities. Through *Cassandra*, the user may select the ap-

propriate dataset(s), preprocess it, and select a DM API to build his/her models (WEKA and R are fully supported, other APIs are partially supported also). Results are displayed and stored, while action can be taken (in an autonomous or semi-autonomous manner), when deemed appropriate.

We explored the use of the ESN and NEAR methodologies as autonomous adaptation methods of reservoir computing topologies, and applied them in order to enhance the prediction ability of our MAS system with respect to short term power load and market price values in the Greek power stock market. In many tasks, non-standard, specially constructed algorithms that usually don't exist in standard DM packages are required. Using the Cassandra system a large and heterogeneous set of DM algorithms could be easily tested in a semi-automated way or be made to autonomously adapt to the problem at hand.

Prediction of load and prices are very important to both the power system administrators, in terms of the economy and system security, as well as the players involved, giving them strategic advantage over their competitors. Our methodology results in very small error for day-ahead power load forecasts, while for price forecasting it outperforms standard data mining algorithms. We believe that with the incorporation of exogenous factors that interfere with market prices, the prediction capability of the system will further improve.

References

1. Amin, M.: Restructuring the Electric Enterprise: Simulating the Evolution of the Electric Power Industry with Intelligent Agents. In: Electricity Pricing in Transition, pp. 27–50. Kluwer Academic Publishers (2002)
2. Amin, M., Ballard, D.: Defining new markets for intelligent agents. IEEE IT Professional 2(4), 29–35 (2000)
3. Babinec, Š., Pospíchal, J.: Optimization of echo state neural networks for electric load forecasting. Neural Network World 2(7), 133–152 (2007)
4. Bell, M.: Service-Oriented Modeling: Service Analysis, Design, and Architecture. John Wiley and Sons, Inc. (2008)
5. Bremer, J., Andressen, S., Rapp, B., Sonnenschein, M., Stadler, M.: A modelling tool for interaction and correlation in demand-side market behavior. In: First European Workshop on Energy Market Modeling Using Agent-Based Computational Economics, Karlsruhe, pp. 77–91 (March 2008)
6. Cao, L., Gorodetsky, V., Mitkas, P.: Agent mining: The synergy of agents and data mining. IEEE Intelligent Systems 24(3), 64–72 (2009)
7. Chatzidimitriou, K.C., Mitkas, P.A.: A NEAT way for evolving echo state networks. In: European Conference on Artificial Intelligence. IOS Press (August 2010)
8. Conzelmann, G., Boyd, G., Koritarov, V., Veselka, T.: Multi-agent power market simulation using emcas. In: IEEE 2005 Power Engineering Society General Meeting, vol. 3, pp. 2829–2834 (2005)
9. D'Inverno, M., Luck, M.: Understanding Agent Systems. Series on Agent Technology. Springer, Heidelberg (2003)
10. Diosteanu, A., Cotfas, L.: Agent based knowledge management solution using ontology, semantic web services and gis. Informatica Economică 13(4), 90–98 (2009)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)

12. He, M., Jennings, N.R., Leung, H.: On agent-mediated electronic commerce. IEEE Transactions on Knowledge and Data Engineering 15(4), 985–1003 (2003)
13. Jaeger, H.: Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network"approach. Tech. Rep. GMD Report 159, German National Research Center for Information Technology (2002), http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNTutorialRev.pdf
14. Lukosevicius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Science Review 3, 127–149 (2009)
15. Maximilien, M., Singh, M.: Agent-based architecture for autonomic web service selection. In: 1st International Workshop on Web Services and Agent Based Engineering, WSABE 2003 (2003)
16. Melzian, R.: Bidding and pricing in electricity markets - agent-based modelling using emsim. In: First European Workshop on Energy Market Modeling using Agent-Based Computational Economics, Karlsruhe, pp. 49–61 (March 2008)
17. Petrov, V., Shebl, G.: Power auctions bid generation with adaptive agents using genetic programming. In: 2000 North American Power Symposium, Waterloo-Ontario, Canada (October 2000)
18. Ponnekanti, S.R., Fox, A.: Sword: A developer toolkit for web service composition. In: 11th International WWW Conference (WWW 2002), Honolulu, HI, USA (2002)
19. Shia, A.: A novel personal agent framework for web services and commercial systems. In: 2006 International Conference on Semantic Web & Web Services, SWWS 2006. CSREA Press, Las Vegas (2006)
20. Showkati, H., Hejazi, A., Elyasi, S.: Short term load forecasting using echo state networks. In: The 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, pp. 1–5 (July 2010)
21. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
22. Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 13–30 (January 2007), http://www.ijcai-07.org/
23. Symeonidis, A.L., Mitkas, P.: Agent Intelligence through Data Mining. Springer, USA (2005)
24. Symeonidis, A.L., Chatzidimitriou, K.C., Athanasiadis, I.N., Mitkas, P.A.: Data mining for agent reasoning: A synergy for training intelligent agents. Engineering Applications of Artificial Intelligence 20(8), 1097–1111 (2007)
25. Tellidou, A., Bakirtzis, A.: Multi-agent reinforcement learning for strategic bidding in power markets. In: 3rd IEEE International Conference on Intelligent Systems, London, UK, pp. 408–413 (September 2006)