27-1-2020

Harvey D. Bastidas C.

INGENI-US eNGINEERING

Singularity

Use case for decentralized trading agent optimization.

# Introduction

This document describes the design and implementation details for an automated forex trading agent that is continuously training in a decentralized network using the singularity platform.

# Problem Description

To find the best way to automate a multi-symbol trading agent while continuously optimizing its performance, a comparison is required between 4 agent automation methods:

* Multi-Timeseries Singular Spectrum Analysis (MSSA) to make a prediction of a trading signal used by the agent.
* NEAT for the dense layer, either using a DCN/LSTM or autoencoder based feature extractor.
* DCN/LSTM and MSSA to predict a trading signal.
* Autoencoder for using its decoder as feature extractor.

The performance ***X*** of an agent in a dataset ***D*** is measured as the net profit ***p*** divided by the risk ***r***.

*X(D) = p / r*

The profit *p* is the variation in the balance of an agent in a forex broker account simulator S after evaluating the market data contained in the dataset D. The risk is a weighted average of all orders individual risk.

The individual order risk is the sum of the instantaneous profit in pips in every tick of the order ~~multiplied by the maximum ticks per order~~ divided by the stop loss ~~multiplied by the length of the order in ticks~~.

***TODO***

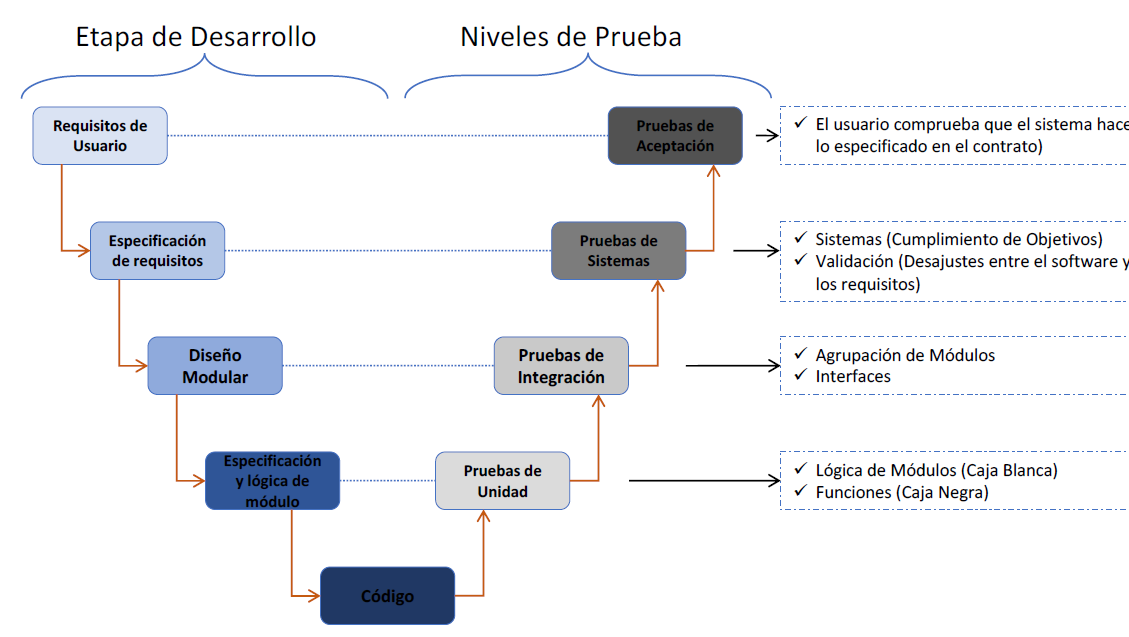
***DEFINIR EN RIESGO INDIVIDUAL mayor cuando SL se hace rápido y menor cuando TP se hace rápido***

# Design Methodology

The design process for the agent is described in this section. This process is based on the Attribute-Driven Design (ADD) iterative design methodology, and the Test-Driven Development (TDD) methodology. The platform design and development process had the following steps:

* Define the stakeholders.
* Define high level architecturally significant requirements.
  1. Define **acceptance tests**.
* Identify the elements that compose the desired system, their behaviors and their interactions.
  1. Define **system tests** (using all components).
* For each of the elements identified:
  1. Identify all the requirements associated with the designed element
  2. Define **integration tests** (for each element’s requirement)
  3. Generate a design solution for the selected element with the following steps:
     1. Select the architectural pattern (AP) that fits the best to its requirement
     2. Select the tactics that are more suitable for implementing the selected AP
     3. Select the technologies used for implementing the selected tactic
     4. Define **unit tests.**
* Setup continuous integration.

The following is a diagram of the described process.



The following is the structure and the lifecycle of the performed tests :

Test case components:

* Input Values
* Pre-conditions in test environment
* Expected results (post-conditions)
* Test execution, instructions or script
* Post-conditions and test results

Test case life cycle:

* Planning and test design
* Data and test environment preparation
* Execution
* Evaluation and log of results
* Acceptation and test end or error notification and request for solution

# Stakeholders

The stakeholders are the users that interact with the program before, during and after development. In this case they are:

* **Agent** **owner**: he wants to maximize his profits by trading with an agent while it is being optimized.
* **Administrator:** configures the optimization process.
* **Developer**: he wants to deliver a scalable maintainable optimization platform.

# User Requirements and Acceptance Tests

This section defines the requirements and acceptance tests for each stakeholder and the acceptance alpha and optionally beta tests they will perform to validate the correct behavior of the developed program. The requirements/tests are divided into functional and non-functional.

## Functional Requirements

The requirements for each stakeholder are shown in this section. The acceptance tests for each requirement are found on Annex 1.

**Agent Owner:**

* Should be able to install the agent and all its requirements for simulation.
* Should be able to configure the agent.
* Should be able to start, pause and stop multiple agents.
* Should be able to generate and download a report detailing the trading operations.

**Developer**:

* Should be able to update the source code.
* Should be able to test the whole program after an update.
* Code should be well-documented.

**Administrator:**

* Should be able to add or remove optimization nodes.
* Should be able to use multiple agent automation methods.
* Should be able to perform multiple optimization processes simultaneously.

## Non-Functional REQUIREMENTS

**Agent Owner:**

* Usability

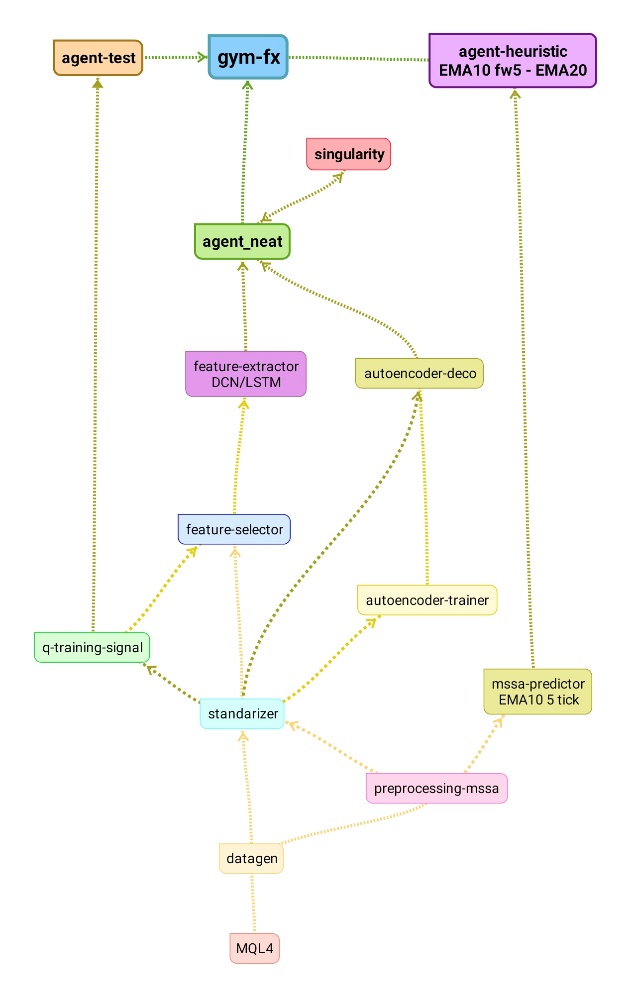
**Developer**:

* Maintainability
* Testability

**Administrator:**

* Scalability
* Fault-Tolerance

# STRUCTURE OF the desired system, and system tests

This section describes the elements that compose the desired system, their behaviors and their interactions. This section does so by enumerate the identified components with a brief description.

The system tests can be found on Annex 1.

* **Gym-fx:**  multi-symbol trading simulator environment.
* **Agent-test**: uses an ideal trading signal to test a training signal.
* **Agent-heuristic**: uses an heuristic MSSA-based strategy to decide the actions in gym-fx.
* **Agent-neat**: uses NEAT to optimize a network that replaces the dense layer of a DCN/LSTM or an autoencoder-based network.
* **Singularity**: extends agent-neat to a decentralized architecture with network event log, allows online evaluation during optimization and management of multiple simultaneous optimization processes.
* **Training-signal:** generate a signal used to test the heuristic strategy EMA 10 fw 5 – EMA 20
* **Feature-extractor:** a DCN and/or LSTM network pre-trained with a training-signal used to reduce the dimensionality o the input feed to a neuro-evolved last layer.
* **Feature-selector:** select the best features to use as input to a network based on a training-signal.
* **Standardizer:** standardize a dataset and generate a file with the standardization parameters.
* **Autoencoder-deco:** load and evaluate a decoder model from a pre-trained autoencoder.
* **Autoencoder-trainer:** train an autoencoder and save the decoder model.
* **MSSA-predictor:** make a prediction using MSSA.
* **Preprocessing-MSSA:** calculate the SSA components and export the MSSA matrix.
* **Datagen:** Generate windowed data per feature.
* **MQL4 data exporter:** generate multiple timeseries in CSV format with prices and technical indicators from an existing Metatrader 4 historic market data.

In the following subsections each one will be described thoroughly, for each of the elements identified:

* Identify all the requirements associated with the designed element
* Define **integration tests** (for each element’s requirement)

## MQL4 data exporter

This section describes high level interaction requirements of a program that generate multiple timeseries in CSV format with prices and technical indicators from an existing Metatrader 4 historic market data.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 2. This component´s requirements are the following:

* Generate timestamped price and technical indicator data and load it from **datagen.**

## DATAGEN

This section describes high level interaction requirements of a program that generate configurable windowed data per feature of input data.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 3. This component´s requirements are the following:

* Generate timestamped price and technical indicator data with MQL4 Data Exporter and load it from **datagen** (same as 6.1)**.**
* Generate standardized data with **standardizer** from datagen utput.
* Generate mssa data with **preprocessing-mssa** from datagen output.

## Standardizer

This section describes high level interaction requirements of a program that standardize a dataset and generate a file with the standardization values.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 4. This component´s requirements are the following:

* Generate standardized data from datagen
* Generate standardized data from preprocessing-mssa
* Use standardized data from feature-selector
* Use standardized data from autoencoder-trainer

## Preprocessing-mssa

This section describes high level interaction requirements of a program that calculate the SSA components and export the MSSA matrix.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 5. el component´s requirements are the following:

* Use data from datagen and import it on pre-processing-mssa
* Generate data for SSA-DCN2D(component, window\_delay)/LSTM feature extractor
* Generate data for direct SSA-NEAT
* Generate data for autoencoder-trainer (one autoencoder/channel?)
* Generate data for mssa-predictor

## Gym-fx

This section describes high level interaction requirements of a multi-symbol trading simulator environment. The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 2. This component´s requirements are the following:

* Load non-normalized price time series from ***MQL data generator*** for agent action execution and load the exact same data as the action time series but windowed and pre-processed to be used as observation matrix from:
  + **training-signal** for agent-test
  + **feature-extractor** for agent-neat
  + **autoencoder-deco** for agent-neat
  + **mssa-predictor** for agent-heuristic
* Use **agent-test** to obtain a performance score and trading event log.
* Use **agent-heuristic** to obtain a performance score and trading event log.
* Use **agent-neat** (reproducible? ) with **DCN/LSTM feature extractor** to obtain a performance score and trading event log.
* Use **agent-neat** (reproducible? ) with **autoencoder** based feature extractor to obtain a performance score and trading event log.

## AGENT-TEST

This section describes high level interaction requirements of a multi-symbol trading agent that uses an ideal trading signal to test a training signal.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 3. This component´s requirements are the following:

* In **gym-fx**, load non-normalized price time series from ***MQL data generator*** for agent action execution and load the exact same data as the action time series but windowed and pre-processed to be used as observation matrix from **training-signal** (same as 6.1).
* Use **gym-fx** to obtain a performance score and trading event log (same as 6.1).

## AGENT-heuristic

This section describes high level interaction requirements of a multi-symbol trading agent that uses an heuristic MSSA-based strategy to decide the actions in **gym-fx**.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 4. This component´s requirements are the following:

* In **gym-fx**, load non-normalized price time series from ***MQL data generator*** for agent action execution and load the exact same data as the action time series but windowed and pre-processed to be used as observation matrix from **mssa-predictor** (same as 6.1).
* Use **gym-fx** to obtain a performance score and trading event log (same as 6.1).

## AGENT-NEAT

This section describes high level interaction requirements of a multi-symbol trading agent that uses uses NEAT to optimize a network that replaces the dense layer of a DCN/LSTM or an autoencoder-based network to decide the actions in **gym-fx**.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 5. This component´s requirements are the following:

* In **gym-fx**, load non-normalized price time series from ***MQL data generator*** for agent action execution and load the exact same data as the action time series but windowed and pre-processed to be used as observation matrix from:
  + **feature-extractor**
  + **autoencoder-deco**
* Use **gym-fx** to obtain a performance score and trading event log (same as 6.1).

## Singularity

This section describes high level interaction requirements of a framework that extends agent-neat to a decentralized architecture with network event log, allowing online evaluation during optimization and management of multiple simultaneous optimization processes.

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 6. This component´s requirements are the following:

* Use **agent-neat** to optimize a dense layer for a **DCN/LSTM** **feature-extractor** network**.**
* Use **agent-neat** to evaluate an optimized dense layer for a **DCN/LSTM** **feature-extractor** network**.**
* Use **agent-neat** to optimize a dense layer for an **autoencoder-deco** network**.**
* Use **agent-neat** to evaluate an optimized dense layer for an **autoencoder-deco** network**.**

## Training-signal

This section describes high level interaction requirements of a data generator that generate a time stamped signal used to test the heuristic strategy:

EMA 10 fw 5 – EMA 20

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 7. This component´s requirements are the following:

* Use **datagen** to generate a **training-signal.**
* Use **agent-test** with the **training-signal** to obtain a performance score and trading event log.
* Use the **training-signal** to configure a **feature-selector.**

## Training-signal

This section describes high level interaction requirements of a data generator that generate a time stamped signal used to test the heuristic strategy:

EMA 10 fw 5 – EMA 20

The integration tests for each interaction requirement can be found on Annex 1. The design and unit tests for this component can be found on Annex 7. This component´s requirements are the following:

* Use **datagen** to generate a **training-signal.**
* Use **agent-test** with the **training-signal** to obtain a performance score and trading event log.
* Use the **training-signal** to configure a **feature-selector.**
* **Training-signal:** generate a timestamped signal used to test the heuristic strategy:

EMA 10 fw 5 – EMA 20

* **Feature-extractor:** a DCN and/or LSTM network pre-trained with a training-signal used to reduce the dimensionality o the input feed to a neuro-evolved last layer.
* **Feature-selector:** select the best features to use as input to a network based on a training-signal.
* **Standarizer:** standardize a dataset and generate a file with the standardization values.
* **Autoencoder-deco:** load and evaluate a decoder model from a pre-trained autoencoder.
* **Autoencoder-trainer:** train an autoencoder and save the decoder model.
* **MSSA-predictor:** make a prediction using MSSA.
* **Preprocessing-MSSA:** calculate the SSA components and export the MSSA matrix.
* **Datagen:** Generate windowed data per feature.
* **MQL4 data generator:** generate multiple timeseries in CSV format with prices and technical indicators from an existing Metatrader 4 historic market data.
* Define high level architecturally significant requirements.
  + Define **acceptance tests**.
* Identify the elements that compose the desired system, their behaviors and their interactions.
  + Define **system tests** (using all components).
* For each of the elements identified:
  + Identify all the requirements associated with the designed element
  + Define **integration tests** (for each element’s requirement)
  + Generate a design solution for the selected element with the following steps:
    - Select the architectural pattern (AP) that fits the best to its requirement
    - Select the tactics that are more suitable for implementing the selected AP
    - Select the technologies used for implementing the selected tactic
    - Define **unit tests.**
* Setup continuous integration.