Introduction to Algorithm As A Service (AAAS)

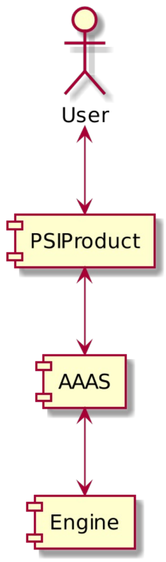
Context of usage and Containers

1. Context of 3AS usages

The most basic Community AIORAE needs:

- use already existing artificial inteligence, data engeneering tools and algorithms. This means using as "The engine": python, R, mathematica, matlab, julia to name a few.
- TODO business needs
- it should be simple to add to the product
- simple to set up (3AS, the engine, monitoring, scaling etc)

AI OR AE Context

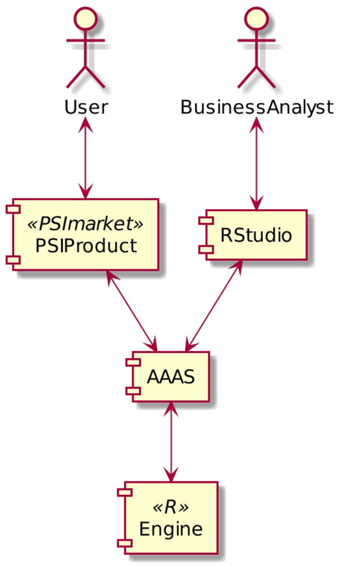


1.1 Context of 3AS in PSImarket

PSImarket needs:

- solve forecasting problems - Time Series based problems
- use R as engine implementation
- provide means for business analysts to work with production data in RStudio
- it should take very limited effor to introduce R to PSImarket

EDM/PSImarket Context



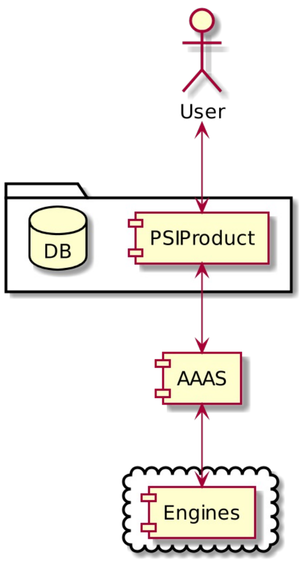
2. Containers

This point mostly opens questions:

- where does the date come from (time series, images, sounds, parametrization)?
  - from product?
  - product data pushed to auxiliary engine database accesible direcltly from the entine?
  - implementation of data access layer in the engine?

- how communication is done?
    - sync/async
    - are the results stored on some intermediate database?
    - what is the deployment strategy?
- where is the source/binary evaluated by the engine
    - how it is defined? on what data? what API - will it be the same as production?
    - how it is provided to the engine? statically? dynamically?
- the same question applies to trained models
- address "openshift issue" (and cloud)?

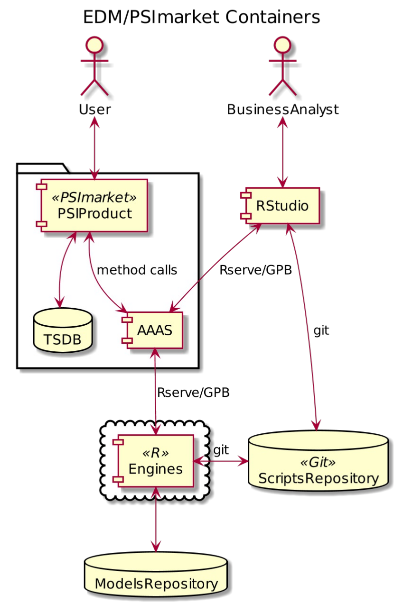
AI OR AE Containers



2.1 EDM/PSImarket Containers

3AS implementation for EDM answers to those questions:

- Data (Time Series) is provided by PSImarket and passed through 3AS to R. Parametrization is passed with calculation request.
- Communication is done synchronously with method calls - library is embedded.
- Scripts are available for both, RStudio and R through Git repository
- Use case from Rstudio works exaclty the same way as Use case from product
- Forecasting models are stored on ModelsRepository to be available in the future.
- Engines, Scripts repository are dockerized, ready to be run on k8s or openshift



3. EDM Components

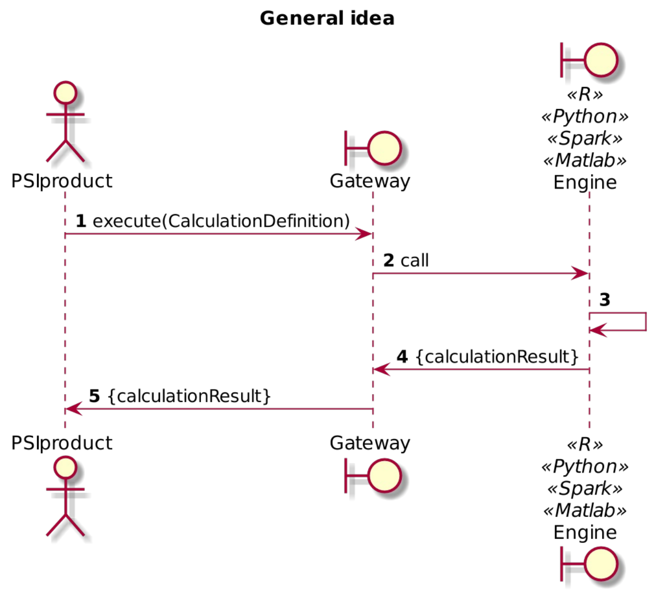
Use cases

Below diagrams describe different use cases on sequence diagrams gradually introducing new components.

1. The most general idea

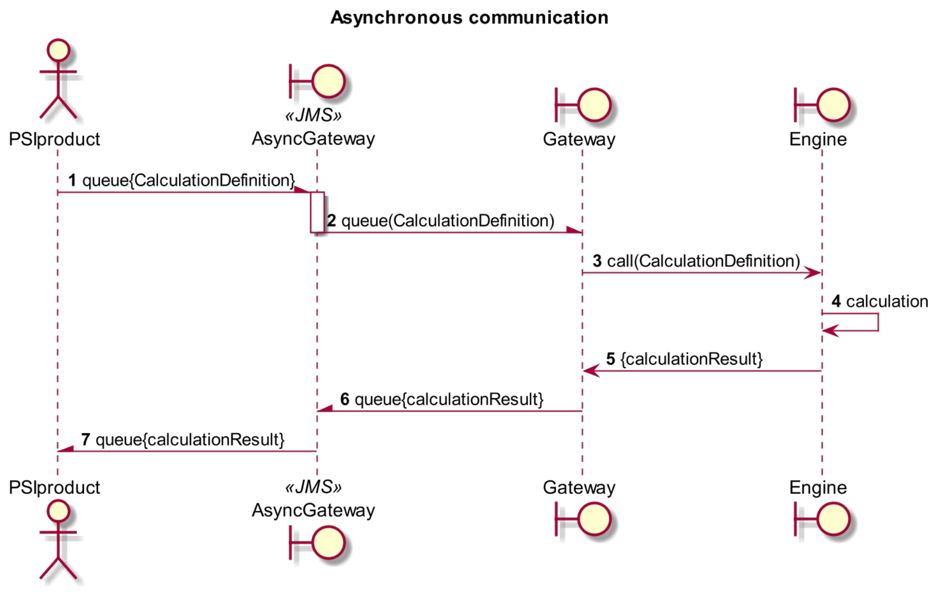
The simplest and most basic form of our requirements is to call some prepared definition on the Engine through gateway/bridge component (thus integration with PSIproduct consumes small amount of effort). Such a general idea can be turn into many particular use cases involving single call of external computational engine, e.g.:

- forecasting/predicting with already prepared forecasting model,
- categorizing the data with respect to trained and available Machine Learning model,
- an application of a unsupervised Machine Learning on provided data.


General idea
PSIproduct
Gateway
«R» «Python» «Spark» «Matlab» Engine
1 execute(CalculationDefinition)
2 call
3
4 {calculationResult}
5 {calculationResult}
PSIproduct
Gateway
«R» «Python» «Spark» «Matlab» Engine

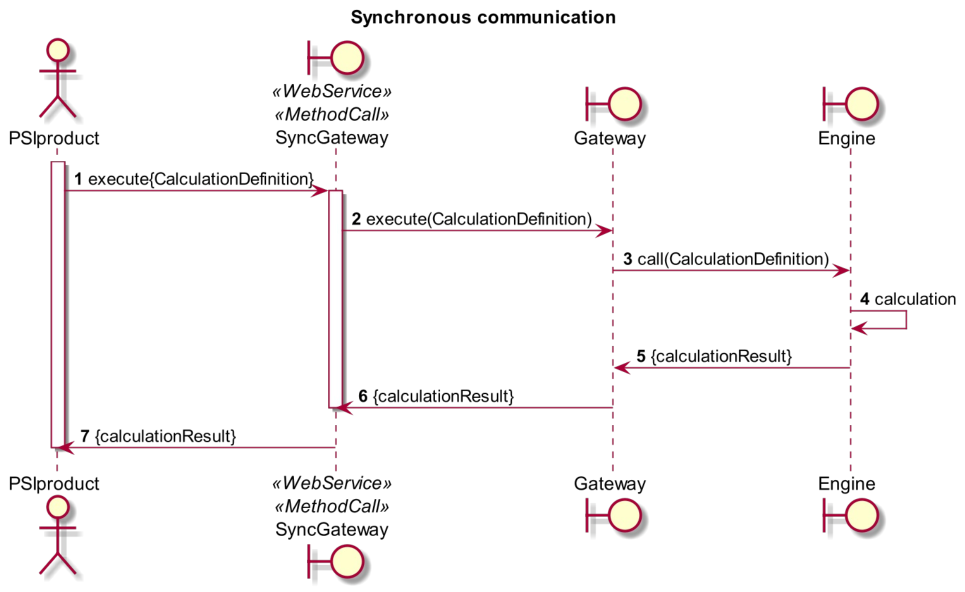
1.1 The algorithm call with asynchronous communication

This kind of architecture is more precise description of general sketch, shown in previous section. The process is asynchronous from the point of view of business user. An additional gateway has been introduced in order to coordinate the flow of computational process. In particular one can imagine an instantiation of multiple computational engines and passing the calculation request to chosen one or multiple ones, the latter in the case of parallel execution.


Asynchronous communication
PSIproduct
«JMS»
AsyncGateway
Gateway
Engine
1 queue{CalculationDefinition}
2 queue(CalculationDefinition)
3 call(CalculationDefinition)
4 calculation
5 {calculationResult}
6 queue{calculationResult}
7 queue{calculationResult}

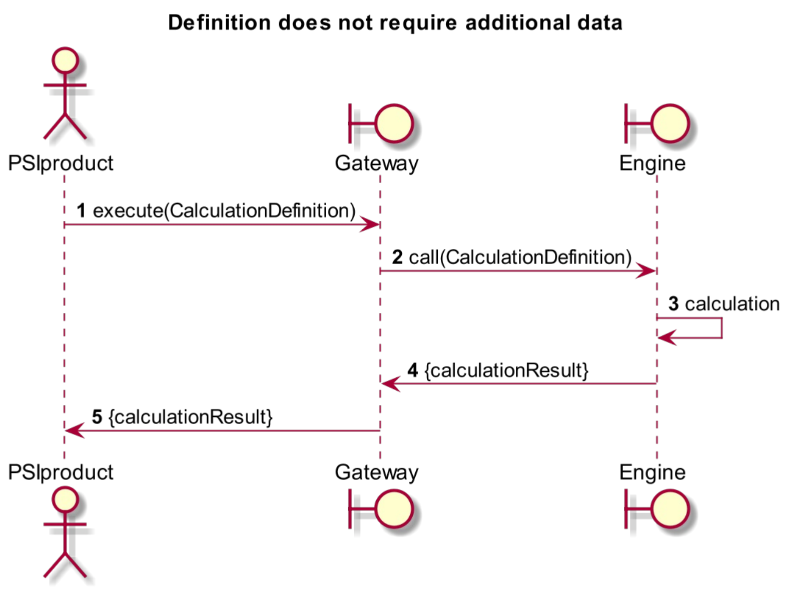
1.2 The algorithm call with synchronous communication

Similar to the previous one, but the communication is synchronous.



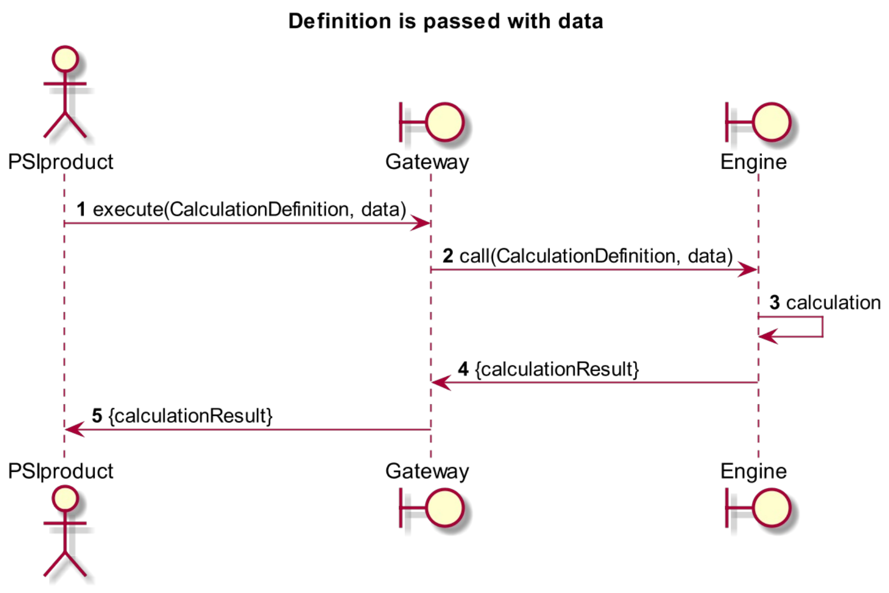
1.3 The algorithm does not require additional data

The scenario where no additional data is required for the calculation


Definition does not require additional data
PSIproduct
Gateway
Engine
1 execute(CalculationDefinition)
2 call(CalculationDefinition)
3 calculation
4 {calculationResult}
5 {calculationResult}
PSIproduct
Gateway
Engine

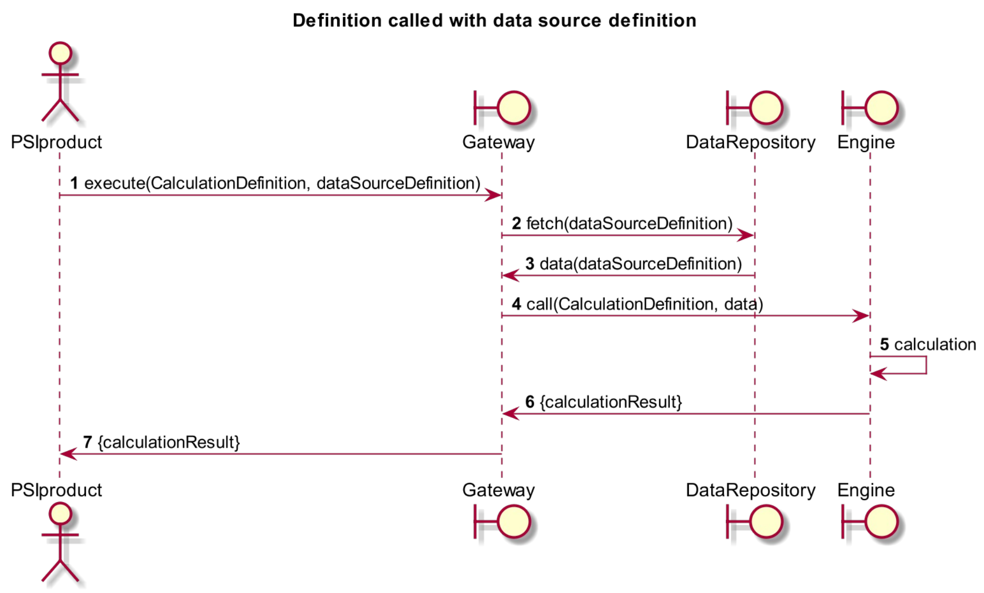
1.4 The required data is passed together with calculation definition

Probably one of the most often scenario. The responsibility of calling application is to provide both the calculation definition and the accompanied data. For instance, one can imagine the forecasting scenario for given period of time (which is a part of calculation definition) where a vector of predictors is also expected (e.g. the forecasted weather conditions time series).


Definition is passed with data
PSIproduct
Gateway
Engine
1 execute(CalculationDefinition, data)
2 call(CalculationDefinition, data)
3 calculation
4 {calculationResult}
5 {calculationResult}
PSIproduct
Gateway
Engine

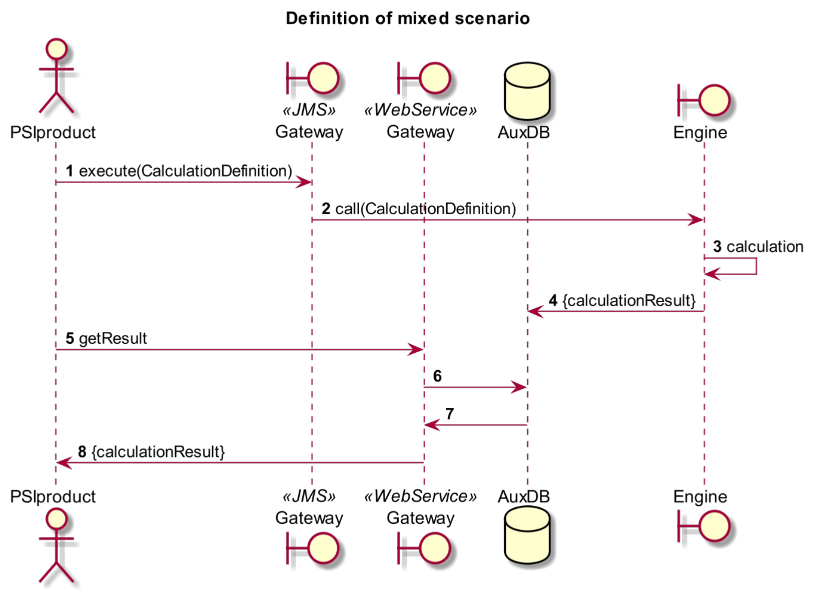
1.5 The required data is fetched from the data base

Scenario where the responsibility of retrieving the data needed by an algorithm is passed to the Gateway. The Gatewey is further passing the data together with the calculation definition to chosen computational engine.


Definition called with data source definition
PSIproduct
Gateway
DataRepository
Engine
1 execute(CalculationDefinition, dataSourceDefinition)
2 fetch(dataSourceDefinition)
3 data(dataSourceDefinition)
4 call(CalculationDefinition, data)
5 calculation
6 {calculationResult}
7 {calculationResult}

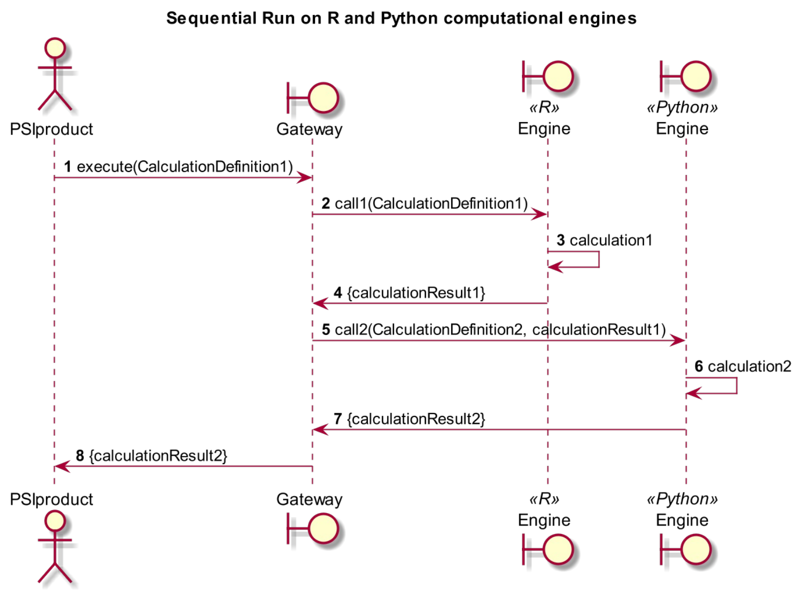
1.6 Mixed scenarion (with auxuliary data base)

Here an additional, auxiliary data base is involved. This data base stores the result of the calculation, which can be further accessed from the application.



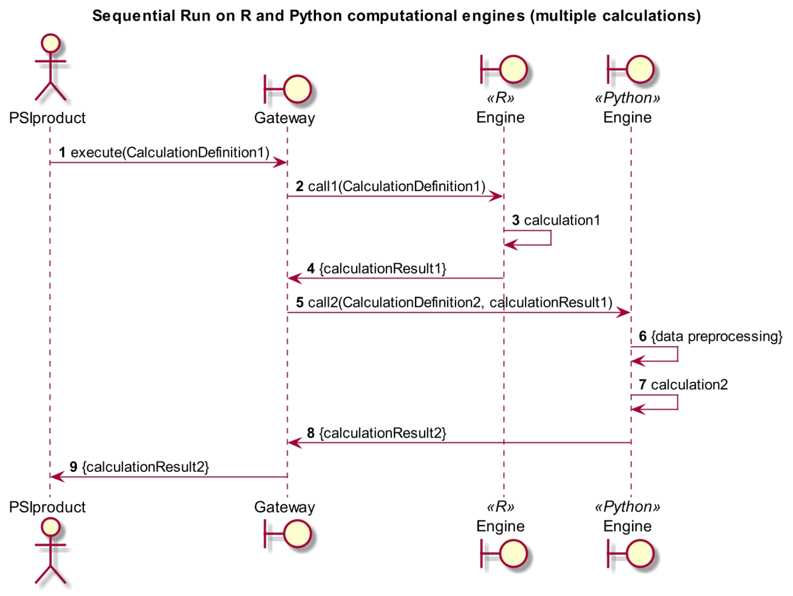
1.7 Sequential run on computational engines

The calculation can be of multistep nature, there fore it can be realised by multiple computational engines. In the example below one can see the R calculation followed by Python execution. In particular the R computational instance can be involved in multicriteria optimization which results in a family of solution forming a Pareto front. The Python computational engine incorporates the user preferences and choses one of Pareto-optimal solutions which is most relevant from the point of view of user preferences.


Sequential Run on R and Python computational engines
PSIproduct
Gateway
«R» Engine
«Python» Engine
1 execute(CalculationDefinition1)
2 call1(CalculationDefinition1)
3 calculation1
4 {calculationResult1}
5 call2(CalculationDefinition2, calculationResult1)
6 calculation2
7 {calculationResult2}
8 {calculationResult2}

1.8 Sequential multiple run on computational engines

Similar to the previous scenarion, there is an additional step carried out within the Python computational engine.


Sequential Run on R and Python computational engines (multiple calculations)
PSIproduct
Gateway
«R» Engine
«Python» Engine
1 execute(CalculationDefinition1)
2 call1(CalculationDefinition1)
3 calculation1
4 {calculationResult1}
5 call2(CalculationDefinition2, calculationResult1)
6 {data preprocessing}
7 calculation2
8 {calculationResult2}
9 {calculationResult2}

1.2 EDM (AKT-1238) use case

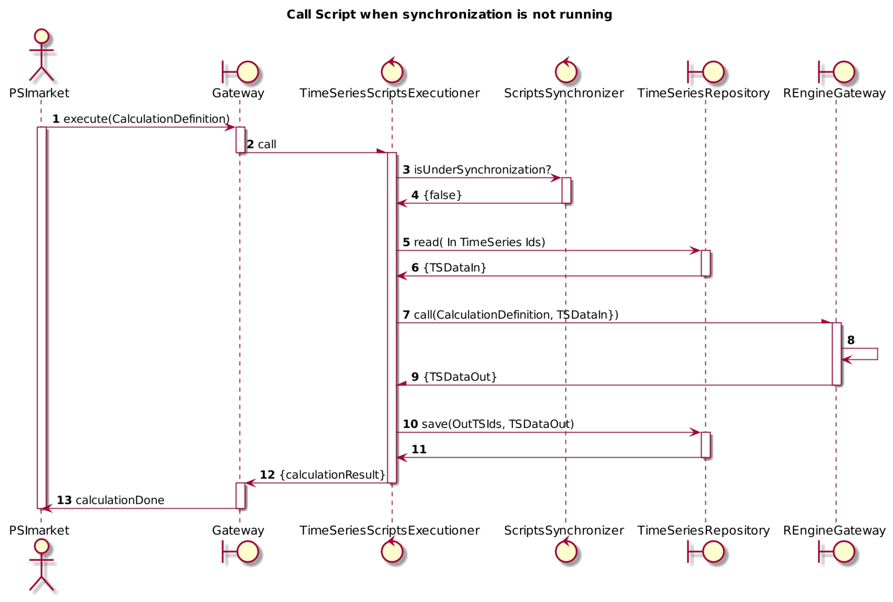
EDM/PSI market use case is Time Series forecasting using R.

R Scripts are developed and pushed to Scripts repository and synchronized with the REngine through ScriptsSynchronizer.

When execution of R script is requested, first scripts synchronization status is checked. Then, according to the definition of time series based calculation, the time series are fetched from PSImarket implementation of TimeSeries Repository and mapped to script variables.

Next step is calling the prognosis script on the engine. When this is finished, time series are returned by

REngine and saved in PSImarket's database using TimeSeriesRepository. After the results are stored in database, PSImarket is informed about result of calculation (error or ok).


Call Script when synchronization is not running
PSImarket
Gateway
TimeSeriesScriptsExecutioner
ScriptsSynchronizer
TimeSeriesRepository
REngineGateway
1 execute(CalculationDefinition)
2 call
3 isUnderSynchronization?
4 {false}
5 read(In TimeSeries Ids)
6 {TSDataIn}
7 call(CalculationDefinition, TSDataIn})
8
9 {TSDataOut}
10 save(OutTSIds, TSDataOut)
11
12 {calculationResult}
13 calculationDone