УДК 681.513.8

ПРОЕКТИРОВАНИЕ СИСТЕМЫ АВТОМАТИЗИРОВАННОЙ СТРУКТУРНО-ПАРАМЕТРИЧЕСКОЙ ИДЕНТИФИКАЦИИ

Андрей Павлов

Международный научно-учебный центр информационных технологий и систем НАНУ и МОНУ, 03680, Киев, пр. Глушкова 40

andriypavlove@gmail.com

У роботі запропоновано сукупність шаблонів проектування програмних додатків мовою UML для розв'язання наступних задач: відділення інтерфейсу програми від моделі даних і ядра системи; абстрагування алгоритму побудови моделей і його складових; абстрагування розрахункової частини системи і відділення її в окрему компоненту; уніфікація процесу розрахунку помилок моделі, її залишків та інших статистичних характеристик.

Ключові слова: об'єктно-орієнтований аналіз, об'єктно-орієнтоване проектування, патерни проектування, MCV, UML, Qt, програмний додаток для моделювання, аналіз даних, система автоматизованої структурно-параметричної ідентифікації.

The paper suggests a set of design patterns to solve the following problems of software development: abstraction of graphic user interface from data model and system's engine; abstraction of the model building algorithm and its components form the other parts of the system's engine; abstraction of the engine form the other parts of the system; unification of model errors calculation process, and processes of calculation of residuals and other statistical characteristics.

Ключевые слова: object oriented analysis, object oriented design, design patterns, MCV, UML, Qt, modeling software, data analysis, automated structure-parametric identification system.

В работе предложено совокупность шаблонов проектирования программных приложений на языке UML для решения следующих задач: разделение интерфейса программы от модели данных и ядра системы; абстрагирование алгоритма построения моделей и его составляющих; абстрагирование расчётной части системы и отделение её в отдельную компоненту; унификация процесса расчёта ошибок модели, её остатков и других статистических характеристик.

Ключевые слова: объектно-ориентированный анализ, объектно-ориентированное проектирование, паттерны проектирования, MCV, UML, Qt, программное приложение для моделирования, анализ данных, система автоматизированной структурно параметрической идентификации.

Введение

При проектировании информационных систем необходимо учитывать множество факторов, влияющих на конечный результат.

Известно, что 70-80% бюджета жизненного цикла системы занимает поддержка системы. Она включает как настройку и адаптацию под конкретного заказчика, так и внесение тактических и стратегических изменений, вызванных изменчивостью условий эксплуатации и окружающей среды. Изменчивость окружающей среды может характеризоваться выходом новой технологии, «отмирание» (прекращение поддержки) используемой технологии,

конкуренцией на рынке программного обеспечения и др. Поэтому сегодня более важной характеристикой системы является не скорость её работы, а скорость внесения в неё изменений. Эта черта системы называется гибкостью и характеризуется простотой внесения изменений и дополнений.

1. Постановка задачи

Гибкость системы непосредственно связана с процессом её проектирования. Наиболее важными и интересными задачами, которые здесь решаются являются следующие две:

- разделение пользовательского интерфейса программы и её движка (engine), позволяющее независимо изменять первое и второе, а также и полностью заменить движок или интерфейс без внесения изменений в остальной код программы;
- добавление в систему нового функционала таким образом, чтобы основной её код не требовал внесения каких-либо изменений.

Для решения этих задач могут применятся так называемые паттерны или шаблоны объектно-ориентированного проектирования [1].

Существуют два подхода к проектированию системы:

- Нисходящий – глобальная задача делиться на множество независимых задач, которые решаются отдельно. Соответственно система делиться на модули, каждый из которых решает конкретную задачу.
- Восходящий – система строиться из элементарных единиц, например, классов, решающих базовые задачи, которые необходимы для достижения сформулированных требований к системе.

Ни один из этих подходов не является оптимальным. При нисходящем подходе можно получить систему, которая в точности не соответствует поставленным требованиям, а в восходящем – систему, которая настолько сильно подогнана под требования, что внесение в неё дополнительных изменений может потребовать перепроектирования системы в целом. Поэтому как правило применяют комбинацию этих подходов: оба подхода применяются одновременно и система проектируется с двух сторон. Этап, где эти два подхода сталкиваются называется интеграцией – процесс объединения всех частей системы в одно целое и отладка проекта.

В данной работе, стоит задача проектирования Системы Автоматизированной Структурно-Параметрической Идентификации (САСПИ), призванную решать такие задачи как прогнозирование, экстраполяцию и аппроксимацию (на основе построения регрессии).

Проектированию системы всегда предшествует этап определения требований.

2. Требования к САСПИ

Сформулируем основные требования к системе:

1. Загрузка матриц данных из текстовых файлов с различным разделителем: «пробел», «табуляция» - *.txt, «,» - *.csv (Excel 2013), «;» - *.csv (Excel 2003).

2. Разбиение выборки:

2.1 Исходных данных на экзаменационную и рабочую подвыборки:

- с последовательным или равномерным разбиением наблюдений;
- с числовым или процентным определением количества наблюдений в выборке экзамена.

2.2 Рабочих данных на обучающую и проверочную подвыборки с числовым или процентным определением количества наблюдений в выборке обучения.

3. Построение моделей:

- прогноза на указанное число шагов вперёд,
- экстраполяции и аппроксимации (регрессии).

При построении задаются:

- Выходная, входные величины. На входные величины могут быть наложены функциональные преобразования такие как: $1/x$, $lag(x)_l$, где l – лаговый сдвиг во времени.
- Внешний критерий:

1) $$NRSS_B = \sum_{i=1}^{n_B} (y_i - \hat{y}_i)^2 \Big/ \sum_{i=1}^{n_B} y_i^2 ,$$

2) $$NRSS_A = \sum_{i=1}^{n_A} (y_i - \hat{y}_i)^2 \Big/ \sum_{i=1}^{n_A} y_i^2 ,$$

3) $$\text{Combi} = (1-\beta)[\alpha \cdot NRSS_B + (1-\alpha)\, NRSS_A] + \beta \cdot |\, NRSS_B - NRSS_A \,| \,/\, |NRSS_B + NRSS_A|, \quad (1)$$

где n_B, n_A – количество наблюдений в выборках проверки и обучения, α – вес ошибки проверки, β – вес несмещённости. Первый и третий критерий используются при экстраполяции, второй – при аппроксимации.

4. Представление результатов моделирования графически и аналитически для каждой из полученных моделей.

5. Анализ результатов.

На выборках экзамена, обучения, проверки и исходной выборке:

- построение гистограмм остатков моделей;
- расчёт математического ожидания и дисперсии остатков;
- расчёт таких известных мер точности модели как: MSE, RMSE, MAPE и др.

– Расчёт информативности каждой из переменных, участвующих в построенных моделях;

6. Создание, редактирование, загрузка и сохранение проектов (сеансов работы с системой).

В данной работе будут рассмотрены вопросы проектирования всех требований, кроме пунктов 4 и 6.

3. Проектирование САСПИ

Существует два подхода к проектированию (и соответственно программированию):

– *структурный* (функционально-ориентированный), использует диаграммы потоков данных (DFD-диаграммы) для моделирования процессов и диаграммы сущность-связь (ERD-диаграммы) для моделирования данных;

– объектно-ориентированный, использует нотацию UML для представления моделей системы.

На сегодняшний день объектно-ориентированный подход к разработке систем является стандартом, и даже требованием при создании хорошо масштабируемых и легко-поддерживаемых систем, поэтому будем использовать именно его.

Рассмотрим первую задачу – проектирование структуры системы таким образом, чтоб её вычислительная часть и данные не зависели от части представления данных, т.е. графического интерфейса пользователя.

3.1 Проектирование модели данных и интерфейса пользователя

Эта задача не является новой. Одно из самых известных решений, которое применяется на практике в среде web-разработки, и является на сегодня основой создания библиотек и фреймворков, это шаблон проектирования *Модель-Контроллер-Вид* (MCV), рис. 1.

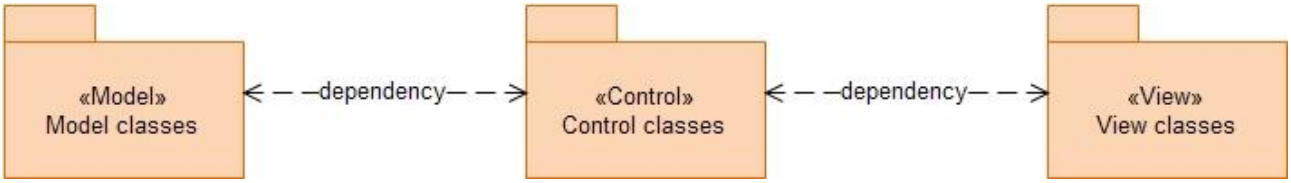

Рисунок 1. Шаблон *Модель-Контроллер-Вид*.

В классическом виде классы пакета *Модель* предоставляют интерфейс для получения и редактирования данных в базе данных, классы *Контроллера* отвечают за обработку событий пользователя при работе с интерфейсом программы, а классы *Вида* ответственны за представление данных пользователю и предоставление ему элементов управления системой.

В нашем случае в требованиях к системе не указано, что она является многопользовательской, а также, очевидно, не нуждается в хранении большого количества данных, поэтому использование базы данных и веб-технологий

здесь не является уместным. Система будет представлять собой настольное приложение под Windows.

При проектировании настольных систем, которые не требуют клиент-серверной архитектуры, обычно классы контроллеров являются избыточными и код обработчиков событий пользователя пишется в классах *Вида*. Поэтому наша исходная задача сводиться к разработке класса шаблонов проектирования типа *Модель-Вид*.

Для реализации интерфейса пользователя будем использовать популярную и широко известную кросс-платформенную библиотеку Qt от Nokia, написанную на C++.

Все классы *Вида* можно разделить на три категории:

- классы, читающие данные *Модели*;
- классы, изменяющие данные *Модели*;
- классы, которые как пишут, так и читаю данные *Модели*.

Идея решения задачи состоит в введении промежуточного класса Link между классами *Вида* и классом *Модели* (рис. 2). Его задача – обновление интерфейса пользователя при изменении *Модели* и обновление *Модели* при вводе новых данных пользователем через интерфейс. Обновление интерфейса осуществляется с помощью абстрактного метода modelChanged, который вызывается из *Модели*, как только нужно обновить *Вид*.

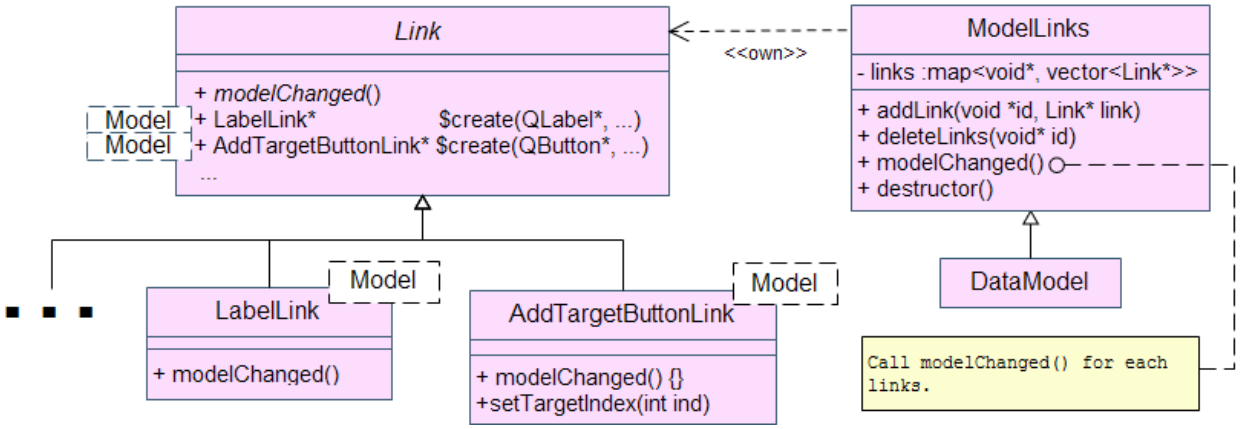

Рисунок 2. Класс Link.

Класс управления связями ModelLinks был отделён от класса модели данных, чтобы логически их разграничить. Ибо основная задача *Модели* (класса DataModel) – это хранить данные и запускать процедуры их обработки. Методы addLink, deleteLink, deleteLinks класса ModelLinks регистрируют и удаляют связи. Для каждого класса *Вида* создаётся свой собственный, производный от базового класса Link, класс связи, который определяет метод modelChanged по своему.

Заметим, что класс Link и производные от него классы являются шаблонными. Параметром шаблона есть *Модель*. Такая конструкция позволяет достичь независимости *Вида* от *Модели* и изменять первое не влияя на второе и наоборот. Т.е. если в будущем потребуется отказаться, например, от

библиотеки Qt, *Модель* и весь движок системы не потребуют изменений. Модификации потребует лишь конкретные классы связей.

3.1.1 Шаблон чтения данных из *Модели*

Рассмотрим класс, читающий данные, LabelLink (рис. 3).

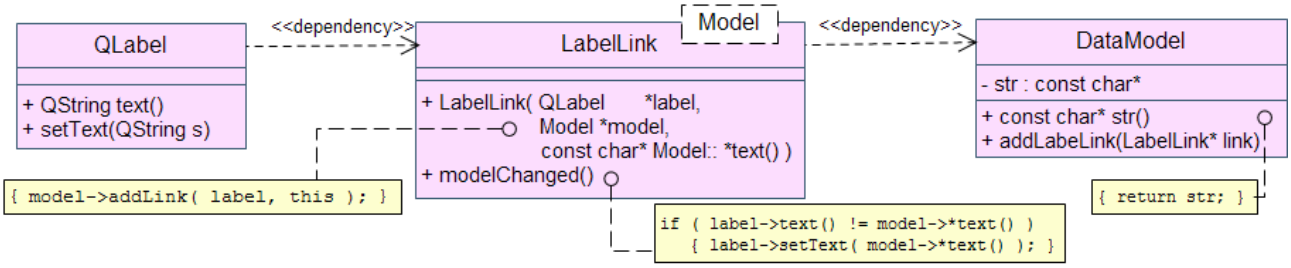

Рисунок 3. Шаблон чтения данных Модели.

QLabel – это класс *Вида* из библиотеки Qt представляющий метку. В конструкторе класса LabelLink мы регистрируем эту связь в *Модели* с помощью метода addLink.

Для удобства использования подклассов класса Link в коде интерфейса системы, класс Link расширен статическими методами создания каждого из производных классов. Таким образом использование в *Виде* выглядит так:

```
QLabel label;
DataModel model;
Link::create( &label, &model, &DataModel::str );
```

3.1.2 Шаблон записи данных в *Модель*

Рассмотрим класс связи AddTargetButtonLink, изменяющий данные *Модели* (рис. 4).

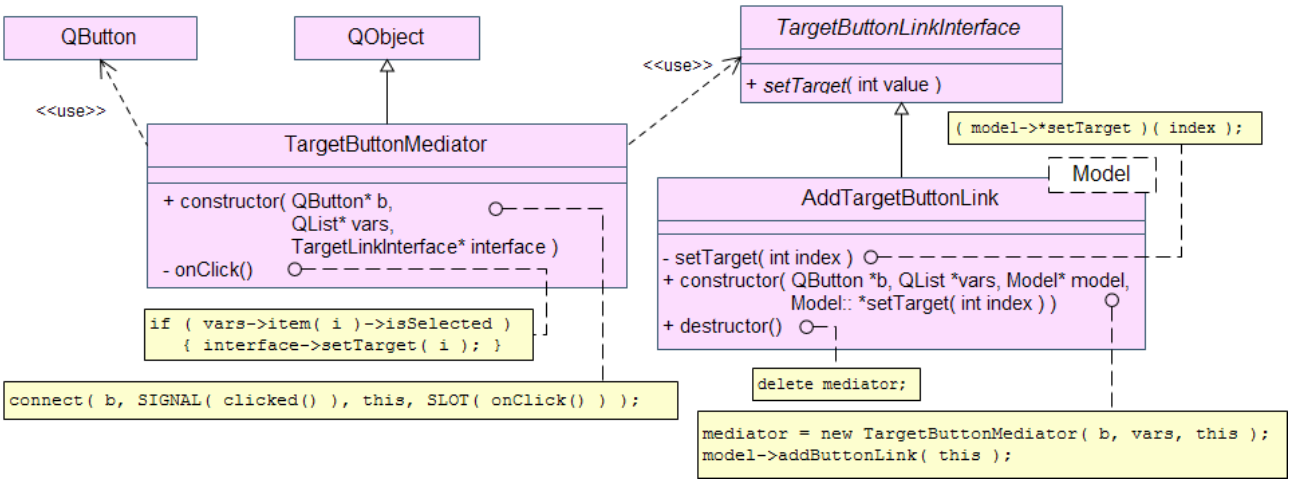

Рисунок 4. Шаблон записи данных в *Модель*.

Для отлавливания и обработки действий пользователя в библиотеке Qt используется механизм сигналов и слотов. Например, сигнал clicked генерируется, когда пользователь нажимает кнопку. Слот – это функция

обработчик события, как правило, сигнала. Метод connect предназначен для связывания любых сигналов и слотов в библиотеке Qt.

Наша задача здесь – создать класс связи ButtonLink, производный QButton, который обрабатывал бы событие её нажатия, и к тому же был шаблонным. К сожалению в библиотеке Qt производный от её классов класс не может быть шаблонным из-за её внутренней архитектуры. Поэтому предлагается ввести класс посредник TargetButtonMediator между классом кнопки QButton и классом связи AddTargetButtonLink. Для того чтоб этот класс мог использовать механизм сигналов и слотов, он должен быть унаследован от класса QObject.

Непосредственно связать класс посредника и связи нельзя, поскольку, в этом случае, первый должен быть шаблонным, поэтому связь осуществляется через абстрактный интерфейс TargetButtonLinkInteface. Такая конструкция, когда для связи двух классов в одном из них выделяется интерфейс, называется шаблоном inversion of control.

Данные которые модифицируют модель задаются в конструкторе класса связи AddTargetButtonLink и передаются посреднику. Объект связи по сути является собственником объекта посредника и удаляет его в своём деструкторе. Обработчик события onClick класса посредника изменяет модель через интерфейс TargetButtonLinkInteface и класс связи AddTargetButtonLink с помощью метода setTarget.

Использование в *Виде* этого шаблона выглядит так:

```
QListWidget  *_varsList;
QToolButton *_addTargetButton;
Link::createAddTarget
(_addTargetButton, _varsList, &_dataModel, &DataModel::setTargetIndex );
```

Единственным недостатком шаблонов для изменения данных является необходимость создания отдельного класса-связи для каждого элемента интерфейса.

3.1.3 Шаблон чтения и записи данных в *Модель*

Рассмотрим класс-связь SpinLink. Объкт QSpin – это элемент управления интерфейса, позволяющий численно задавать (изменять) некоторый параметр *Модели*. В данном шаблоне комбинируются два подхода описанных выше (рис. 5).

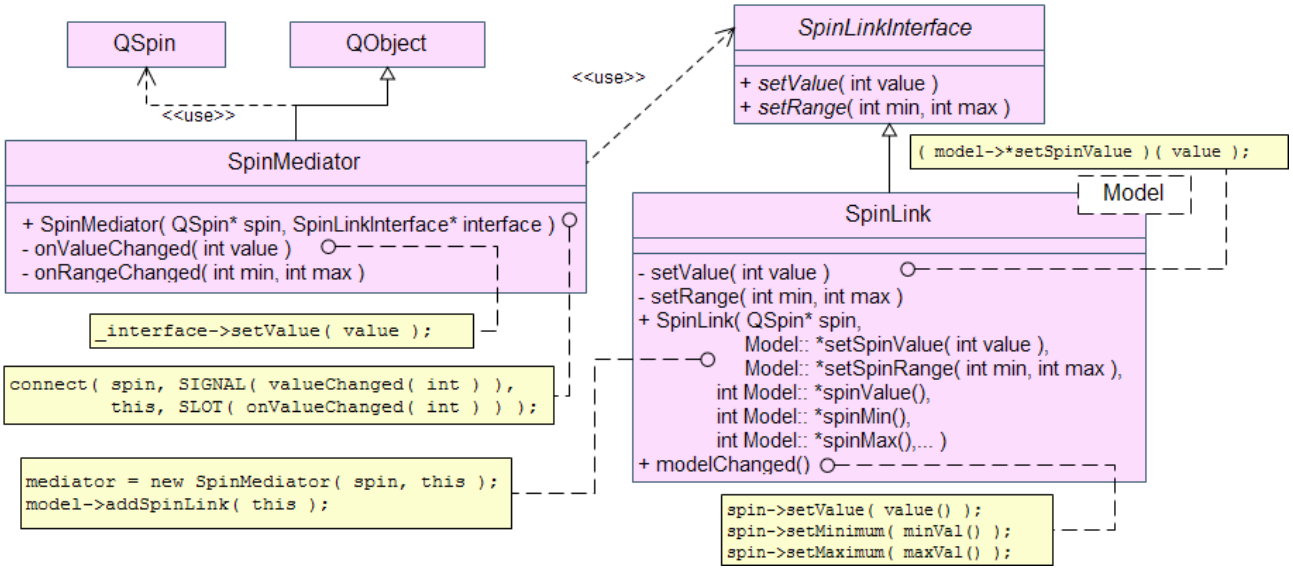

Рисунок 5. Шаблон чтения/записи данных в *Модель*.

Данные для изменения *Вида* читаются из *Модели* методами spinValue, spinMin, spinMax (в методе modelChanged вызываются соответствующие функции value, minVal, maxVal). Запись данных в модель осуществляется по цепочке вызова методов valueChanged (*Вид*) -> onValueChanged (посредник) -> setValue (интерфейс связи) -> setValue (связь) -> setSpinValue (*Модель*).

Использование в *Виде* выглядит так:

```
Link::create( _forecast_spin, &_dataModel, &_dataModel,
              &DataModel::setForecastSteps, &DataModel::setForecastStepsRange,
              &DataModel::forecastSteps, &DataModel::minForecastSteps,
              &DataModel::maxForecastSteps );
```

3.2 Проектирование ядра системы

Ядро системы представляет собой часть системы, ответственную за:

– предварительную обработку данных:
  - разбиение выборки
  - преобразование входных переменных (лаг, обратная величина, перемножение переменных);

– построение моделей прогноза, экстраполяции и аппроксимации;

– постобработку результатов моделирования (расчёт остатков и ошибок модели: MSE, RMSE, MAPE и др.).

Рассмотрим часть ядра, отвечающую за построение моделей. Для построения моделей САСПИ использует обобщённый релаксационный итерационный алгоритм (ОРИА) МГУА [2].

3.2.1 Проектирование ОРИА

В [3] утверждается, что любой алгоритм МГУА состоит из четырёх компонент: генератор структур моделей, метод оценивания параметров, критерий селекции и класс моделей. В [2] отмечается, что компонент *класс*

моделей не требуется для определения итерационного алгоритма МГУА, поскольку процедура формирования конкретного класса моделей является предварительным преобразованием исходных векторов. Поэтому данный компонент входит в часть системы, ответственную за предварительную обработку данных.

ОРИА на сегодня имеет два генератора структур: с направленным и полным перебором аргументов на каждой итерации. Алгоритм использующий генератор полного перебора достаточно прост:

1. Сгенерировать структуры моделей текущей итерации.
2. Оценить параметры этих моделей.
3. Вычислить критерий селекции для построенных моделей.
4. Выбрать множество моделей, которое будет передано на следующую итерацию алгоритма.
5. Проверить правило останова алгоритма. Если оно не выполнено, то перейти на следующую итерацию и повторить на ней шаги 1-5.

Выполнив объектно-ориентированный анализ изложенного выше алгоритма можно выделить пять независимых компонент:

– генератор структур;

– алгоритм оценивания параметров;

– алгоритм расчёта критерия селекции;

– правило отбора моделей;

– правило останова алгоритма.

Эти компоненты могут быть представлены соответствующими абстрактными классами, а алгоритм в целом – взят за базовый многорядный алгоритм МГУА (здесь и далее понятие *многорядный алгоритм* охватывает классы итерационных и многоэтапных алгоритмов). В зависимости от специфических условий, алгоритм может быть конкретизирован за счёт подстановки в него производных классов, которые реализуют соответствующие базовые абстрактные классы. Такой шаблон проектирования называется *стратегией* [1].

Конкретизируем базовый многорядный алгоритм. В нашем случае генератор полного перебора ОРИА эквивалентен генератору упрощённого алгоритма МГУА [4], алгоритм оценивания параметров – это рекуррентный алгоритм из [5]. Обратившись к формуле комбинированного критерия (3) можно заметить, что она состоит из формул (1) и (2) и позволяет при определённых значениях параметров α и β получить критерии (1) и (2). Поэтому нет необходимости в создании абстрактного класса для расчёта критерия селекции – можно обойтись одним классом, реализующим комбинированный критерий.

Алгоритм, использующий генератор с направленным перебором не вписывается в структуру базового многорядного алгоритма, поскольку его процесс генерации структур моделей (п.1 алгоритма) выполняет этапы оценивания параметров, расчёта критерия селекции и отбора моделей по

специальному правилу. Однако, конкретика пяти выделенных выше компонент в данном алгоритме аналогична базовому.

Диаграмма классов для алгоритмов с использованием полного и направленного перебора представлена на рис. 6.

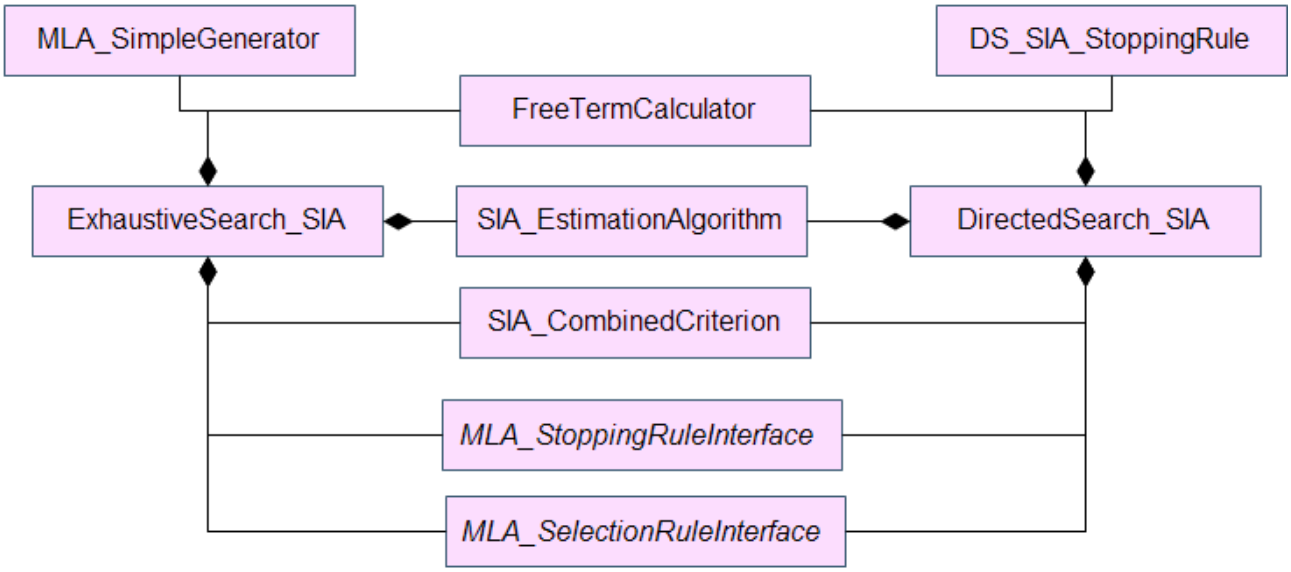

Рисунок 6. Алгоритмы ОРИА и их составляющие.

Алгоритмы полного и направленного перебора представлены классами ExhaustiveSearch_SIA и DirectedSearch_SIA соответственно. Аббревиатуры SIA и MLA означают Simplified Iterative Algorithm и Multi-Layered Algorithm соответственно. Причём, второе понятие является более широким, чем первое. Класс FreeTermCalculator используется для расчёта свободного члена моделей, полученных в конце работы алгоритмов. Класс DS_SIA_StoppingRule реализует специальное правило останова для алгоритма с направленным перебором. Алгоритмы также владеют абстрактными классами:

− правило останова MLA_StoppingRuleInterface

− правило селекции моделей MLA_SelectionRuleInterface,

которые определяют интерфейсы для работы с любыми возможными правилами многорядных алгоритмов. Диаграмма производных классов для интерфейсов правил представлена на рис. 8.

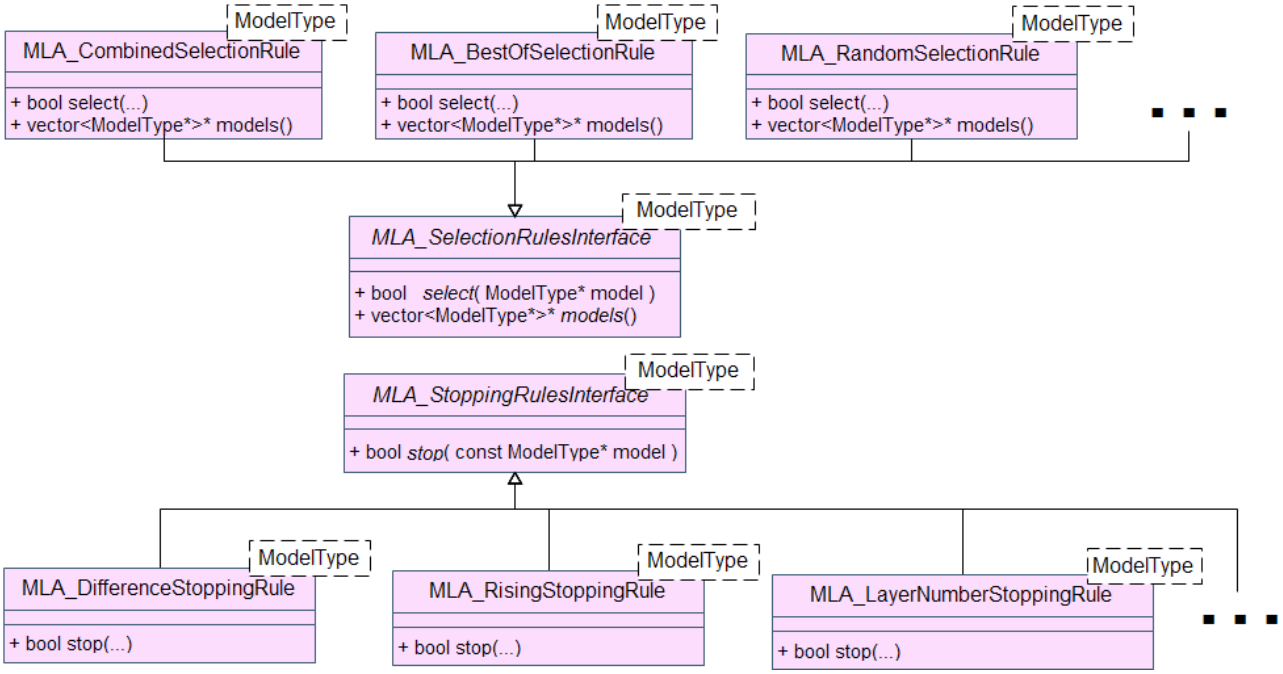

Рисунок 8. Возможные правила отбора и останова в многорядных алгоритмах САСПИ

Как видим, предполагается реализовать три правила отбора моделей:

– MLA_BestOfSelectionRule – отбирает лучшие по заданному критерию модели;

– MLA_RandomSelectionRule – отбирает модели случайным образом;

– MLA_CombinedSelectionRule – комбинирует два выше правила таким образом, что можно задать процент моделей, которые будут выбраны по одному из правил.

Многорядные алгоритмы САСПИ должны уметь останавливаться по следующим правилам:

– MLA_DifferenceStoppingRule – останавливает алгоритм, если разница между значениями критерия лучших моделей соседних рядов меньше заданного порога;

– MLA_RisingStoppingRule – останавливает алгоритм, если на текущем ряде не получена модель лучше чем на предыдущем;

– MLA_LayerNumberStoppingRule – останавливает алгоритм, когда достигнут указанный номер ряда.

Параметром шаблона классов правил является тип обрабатываемой модели. Диаграмма иерархии классов моделей, а также базовых классов компонент алгоритмов, которые с ними могут работать представлены на рис. 9.

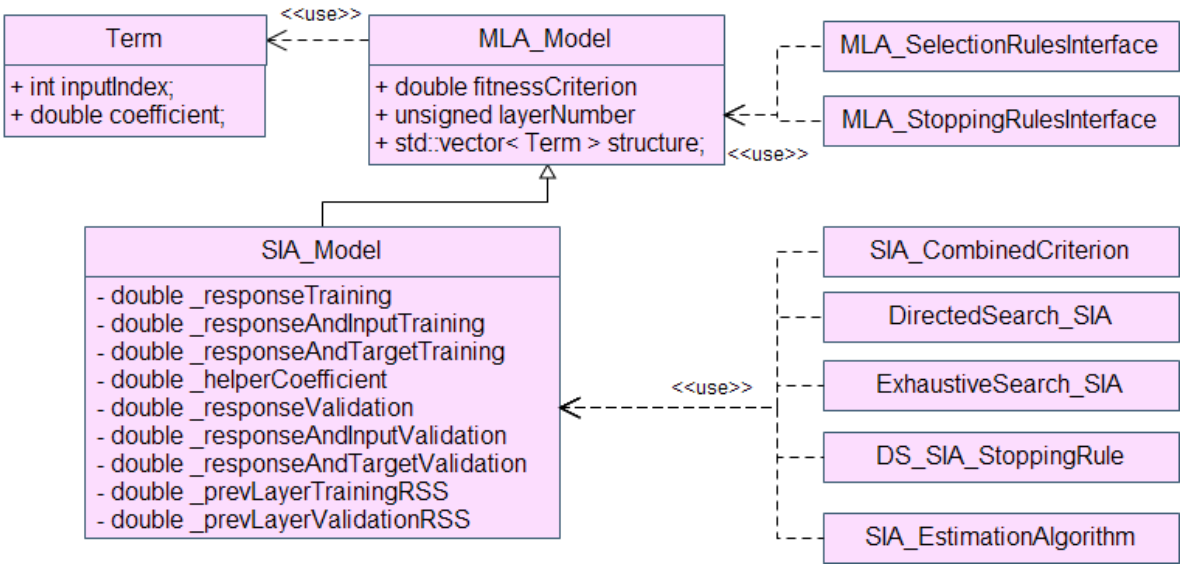

Рисунок 9. Классы моделей и используемые их классы системы

Напомним, что классы с префиксом MLA являются более абстрактными, чем классы с префиксом SIA, поэтому модели упрощённого итерационного алгоритма (класс SIA_Model) могут использоваться только в классах с префиксом SIA, а модели многорядного алгоритма (класс MLA_Model) могут применяться как в классах с префиксом SIA, так и в классах с префиксом MLA.

Отметим, что информация о модели в многорядном алгоритме описывается данными о значении критерия селекции (fitnessCriterion), номером ряда (layerNumber) и массивом одночленов типа Term, представляющих модель линейной регрессии. Класс SIA_Model расширяет эти данные, добавляя к ним вспомогательную информацию, необходимую для рекуррентного расчёта параметров и критерия селекции модели.

С целью удобного использования части ядра системы, ответственную за построение моделей были выделены два класса (рис. 10):

– класс MLA_ManagerInterface, управляющий процессом работы с алгоритмами ОРИА и класс Façade;

– реализующим конструирование конкретного алгоритма ОРИА.

Такой шаблон проектирования называется *фасад* [1].

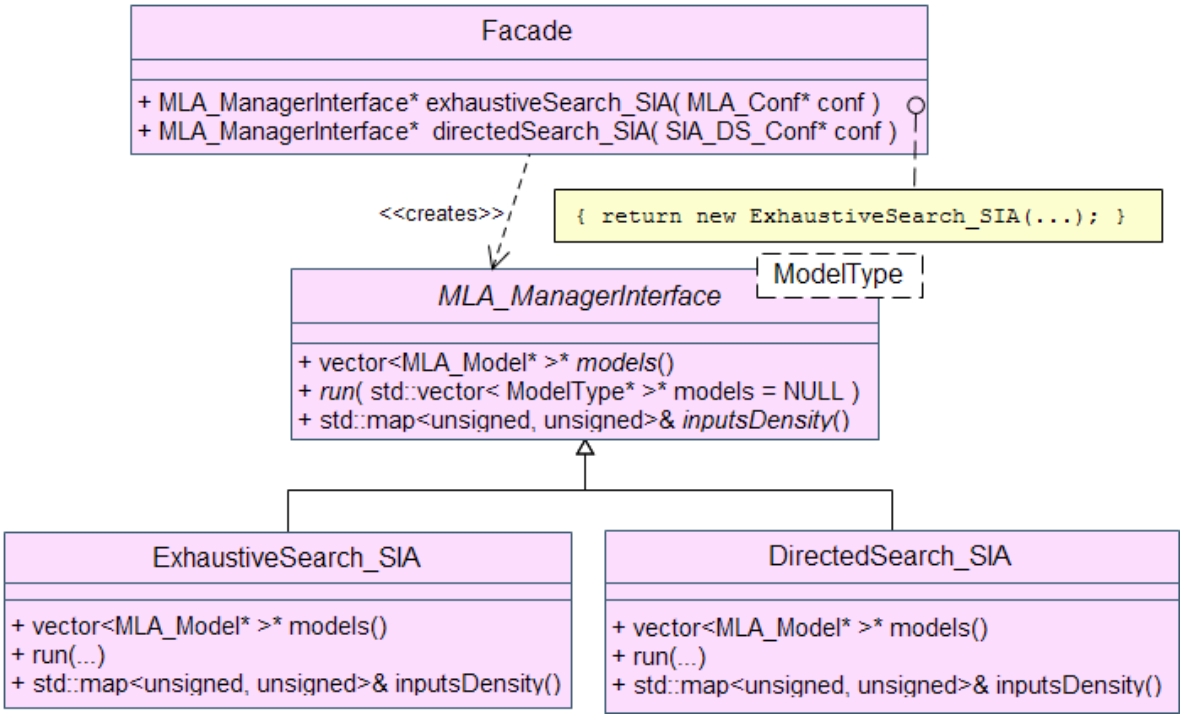

Рисунок 10. Фасад ОРИА

Абстрактный класс MLA_ManagerInterface позволяет:

- получать массив построенных моделей (метод models),
- запускать алгоритм (метод run),
- получать информацию о информативности аргументов (метод inputsDensity)

независимо от того, какой многорядный алгоритм используется. Этот класс был спроектирован так, чтоб в будущем можно было останавливать и продолжать работу алгоритма с указанного места. Поэтому в методе run имеется необязательный параметр – массив построенных моделей, который используется для как отправная точка для расчёта. Поскольку неизвестно, какой алгоритм будет запускаться с этим параметром, необходимо было шаблонизировать класс MLA_ManagerInterface типом используемой модели.

Класс Façade выступает в роли внешнего интерфейса скрывающего за собой все классы ядра, ответственные за построение моделей. Данный класс позволяет создать нужный алгоритм ОРИА, передав ему объект конфигуратор, хранящий такие настройки как: критерии селекции, правило останова, количество лучших моделей, массив протекцируемых переменных и др.

3.2.2 Проектирование предварительной обработки данных

Согласно сформулированным требованиям, предварительная обработка исходной матрицы данных включает как обработку строк матрицы - алгоритмы разбиения выборки данных, так и обработку столбцов – формирование класса моделей, т.е. такие преобразование входных переменных, как взятие лага,

обратной величины и перемножение переменных. На рис. 11 представлена диаграмма классов, реализующими предварительную обработку данных.

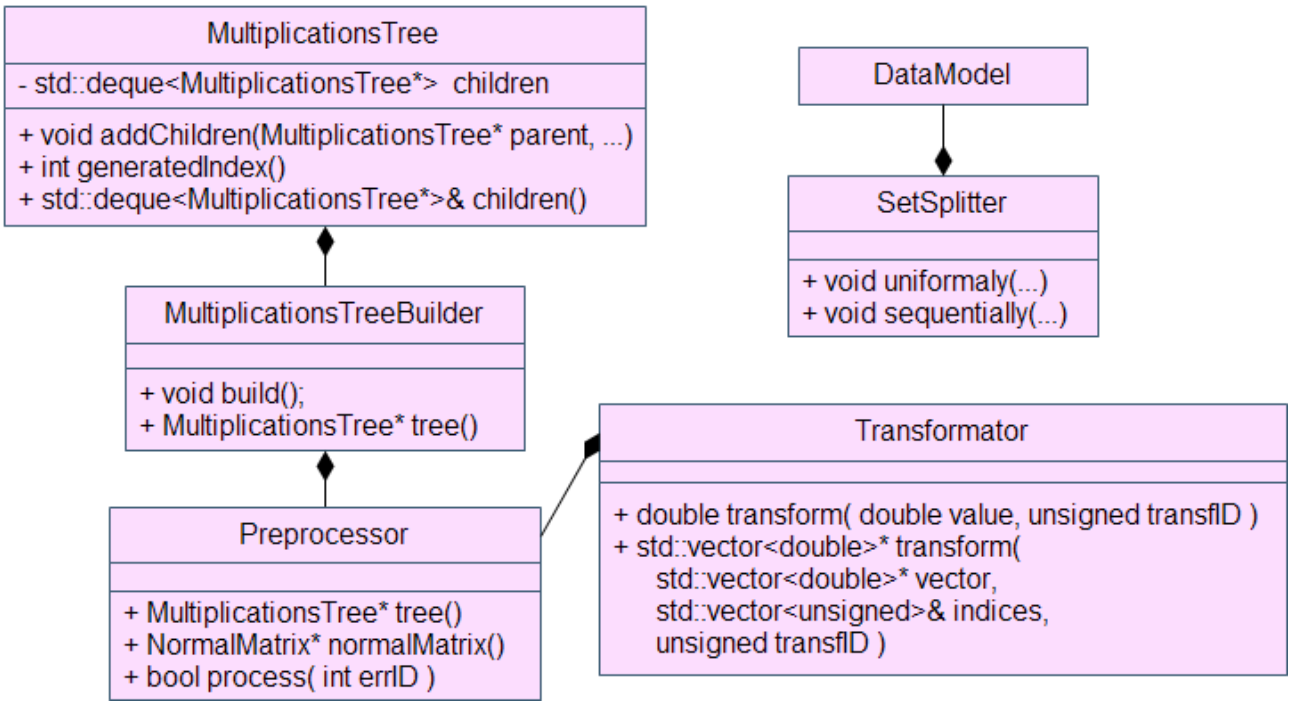

Рисунок 11. Диаграмма классов предварительной обработки данных.

Процесс обработки начинается, когда модель данных вызывает метод uniformaly() или sequentialy() объекта класса SetSplitter чтоб разбить выборку данных с помощью соответствующего алгоритма разбиения. Класс Preprocessor используется для:

– Преобразования входных векторов переменных в вектора состоящих из обратных величин с помощью метода transform() класса Transformator, задав номер трансформации transfID.

– Создания новых входных векторов, отвечающих перемноженным входным переменным с помощью метода build() класса MultiplicationsTreeBuilder.

– Создания матрицы нормальных уравнений, отвечающей всем преобразованным входным переменным.

Далее, полученная нормальная матрица используется алгоритмами ОРИА при оценивании параметров, расчёте критерия селекции и расчёте свободного члена моделей.

3.2.3 Проектирование постобработки данных

Целью постобработки данных является анализ точности построенных моделей с помощью расчёта таких характеристик как:

– ошибки MSE, MAPE, RMSE и др.,

–

– математического ожидания и среднеквадратического отклонения остатков модели,

– расчёт гистограмм остатков

на выборках обучения, экзамена, проверки и общей выборке.

Диаграмма классов, реализующих постобработку представлена на рис. 12.

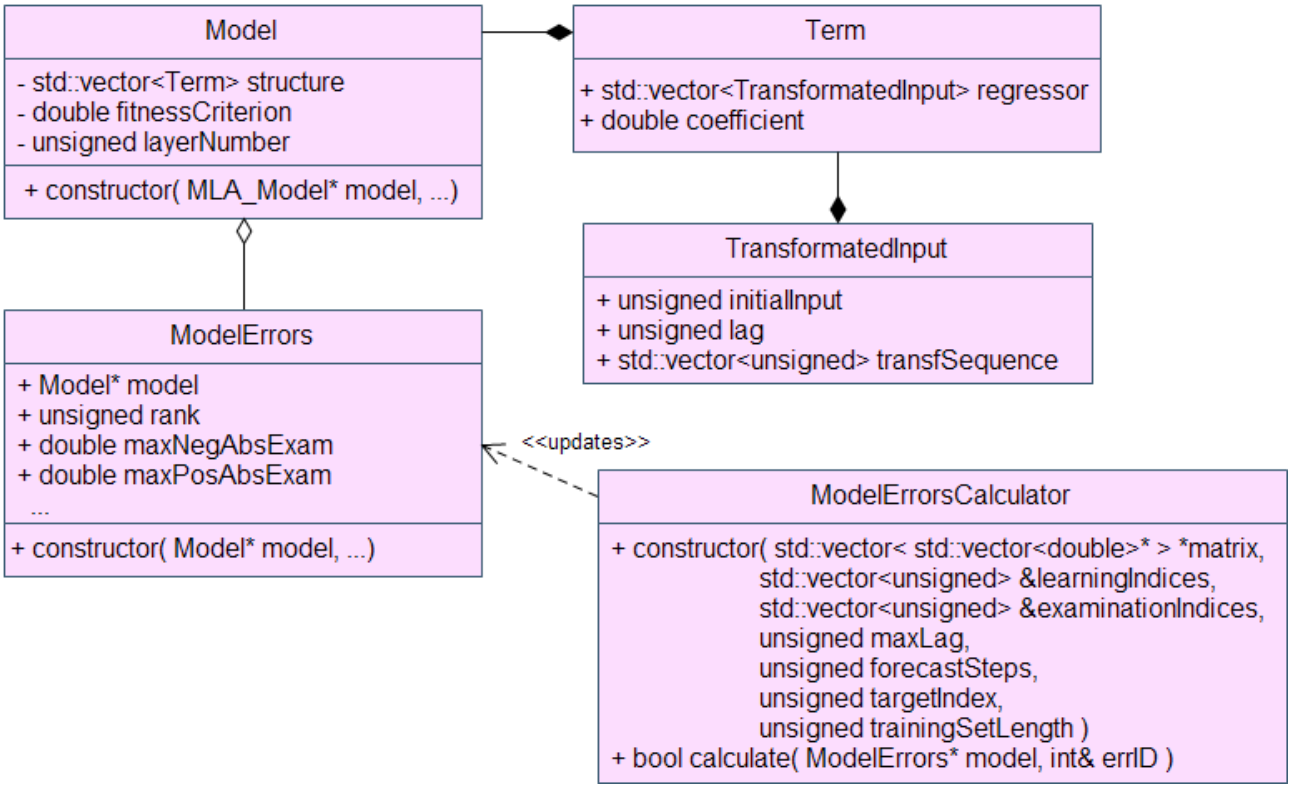

Рисунок 12. Проектирование постобработки данных.

С целью унификации расчёта характеристик моделей полученных по различным алгоритмам, был введен класс Model. На данный момент система работает с только с одним базовым классом MLA_Model, поэтому пока что имеется лишь один конструктор класса Model. Если в систему в будущем будут добавлены алгоритмы полного перебора, необходимо будет добавить соответствующий конструктор для класса Model. Структура модели определяется массивом одночленов (класс Term). Одночлен состоит из регрессора и коэффициента. Регрессор в сою очередь, это массив преобразованных исходных переменных (класс TransformatedInput), которые должны быть перемножены.

Дабы отделить модель и данные связанные с её статистическими и точностными характеристиками, введен класс ModelErrors. Класс ModelErrorsCalculator рассчитывает все характеристики модели основываясь на входной матрице, индексах рабочеё и экзаменационной последовательностях, максимальном лаге, количестве шагов прогноза, выходной переменной и длине обучающей последовательности.

Для расчёта гистограмм ModelErrors хранит массив остатков модели и длина интервала гистограммы на общей, рабочей и экзаменационной выборках.

При построении гистограмм предполагается нулевое математическое ожидание остатка модели.

3.3 Проектирование динамического аспекта работы системы

Проектирование информационных систем охватывает множество взглядов на систему с разных сторон: статический вид, описывающий данные и их взаимосвязи; динамический вид, отражающий взаимодействию классов и компонент системы во времени; вид развёртывания системы, описывающий распределение компонент системы по физическим вычислителям (компьютерам); и др. Самыми важными являются первые два.

Предыдущие разделы описывали статическую сторону системы с помощью диаграмм классов, в этом подразделе сосредоточимся на динамическом аспекте, и опишем процесс построение модели.

Поскольку полная диаграмма последовательности слишком большая, она разделена на две части и представлена на рисунках 13, 14.

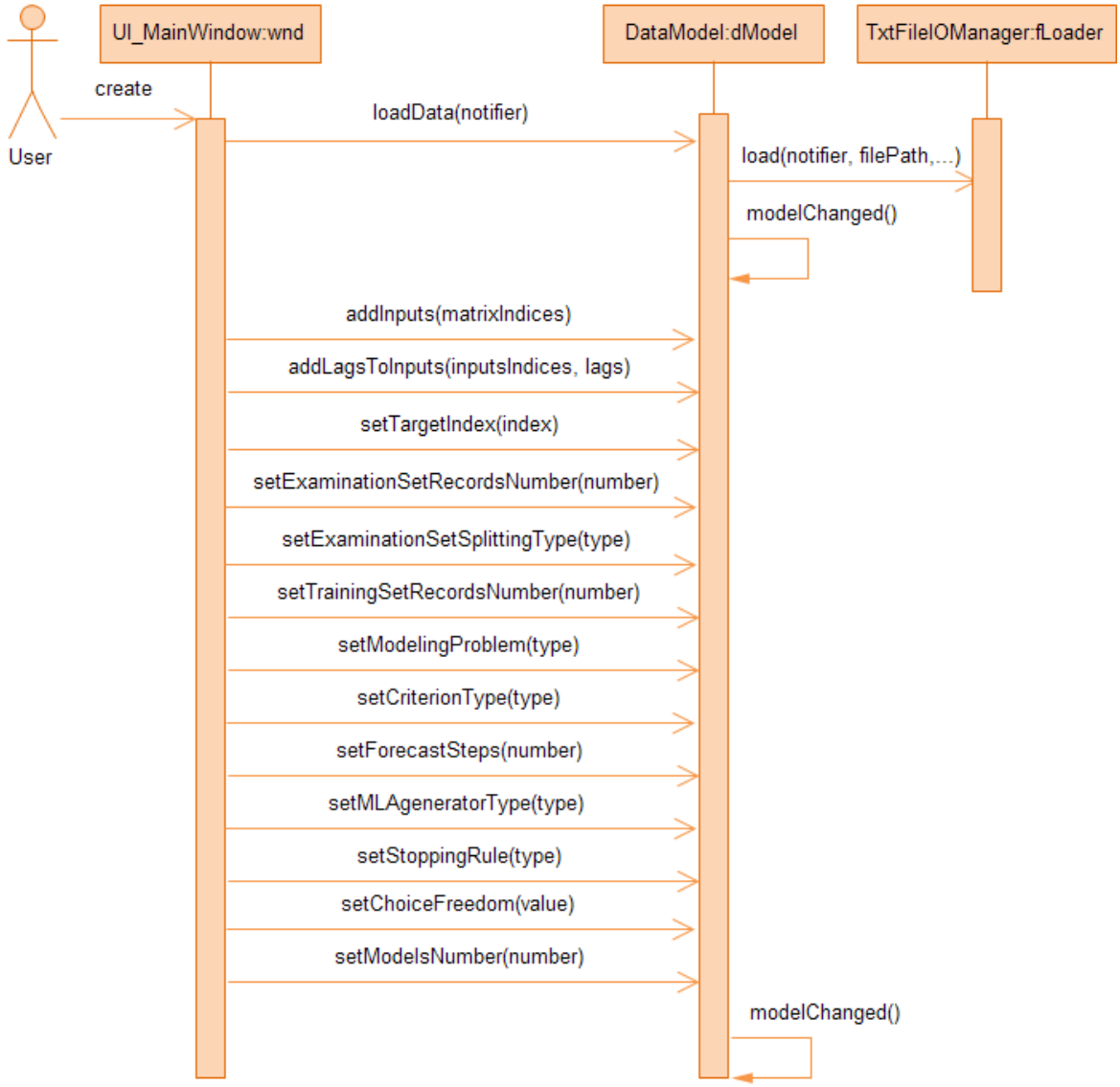

Рисунок 13. Процесс построение модели часть 1: загрузка данных, установка параметров моделирования

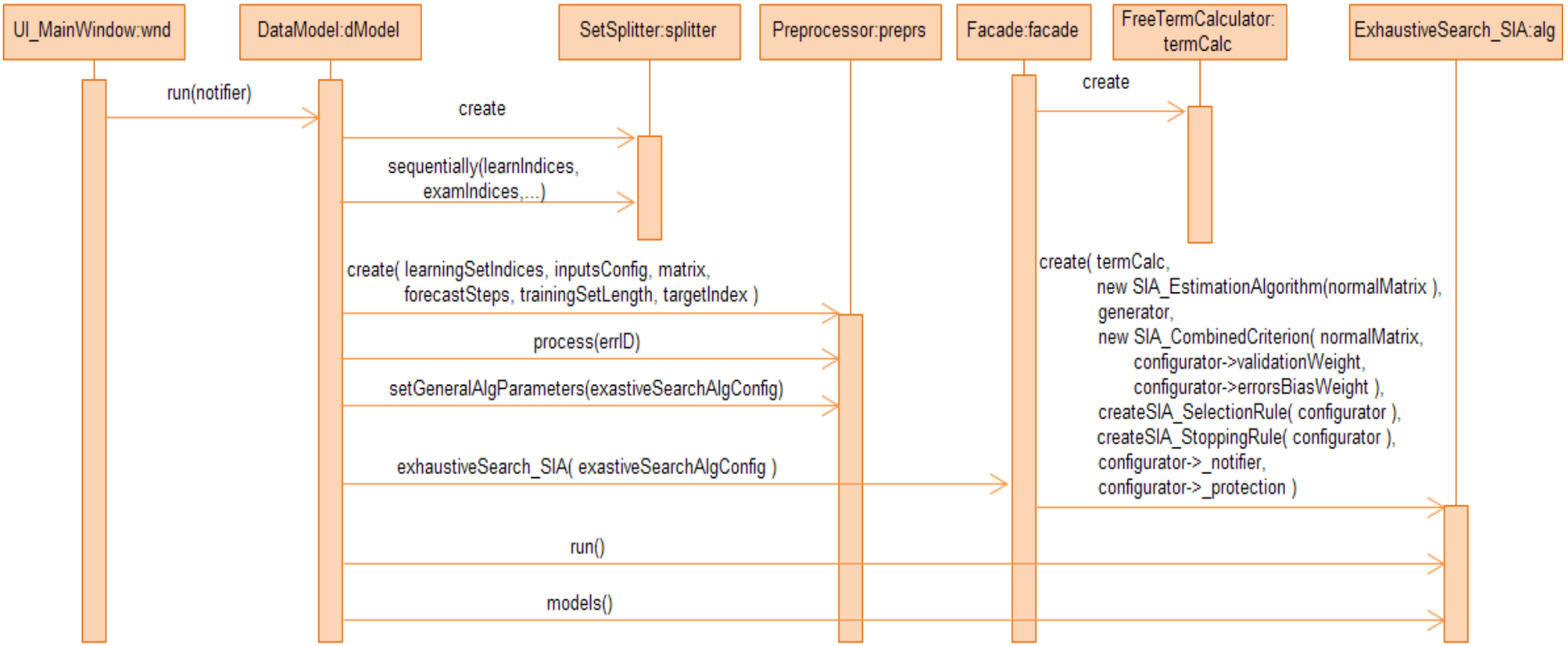

Рисунок 14. Процесс построение модели часть 2: построение модели

4 Выводы

В работе решены несколько прикладных актуальных технических задач проектирования компьютерных информационных систем, а именно:

– создан класс шаблонов проектирования, позволяющих полностью заменить интерфейс системы не внося изменений при этом в остальную часть системы;

– ядро системы спроектировано таким образом, что позволяет легко и быстро добавить или модифицировать алгоритмы построения моделей и их составляющие;

– выделен интерфейс абстрагирующий расчётную часть системы, что позволяет использовать её в любых других информационных системах, требующих построения моделей линейной регрессии;

– модель проекта предварительной обработки данных позволяет легко расширять состав функций-преобразований, внося изменения только в один класс;

– для постобработки данных выделен унифицированный класс Model, над которым осуществляется алгоритм обработки результатов моделирования; для обработки результатов моделирования моделей, полученных различными алгоритмами, необходимо добавить только конструктор для класса Model, конструирующий из полученной модели объект класса Model.

Литература

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software // Addison-Wesley. 1994. P. 395 ISBN 0-201-63361-2.

2. Павлов А. В. Обобщённый релаксационный итерационный алгоритм МГУА // Індуктивне моделювання складних систем. Зб. наук. праць, вип. 2. – К.: МННЦІТС НАНУ, 2011. – С. 95-108.

3. Степашко В.С. Алгоритмы МГУА как основа автоматизации процесса моделирования по экспериментальным данным / Автоматика. – № 4. – С. 44-55.

4. Шелудько О.И. Самоорганизация математических моделей при решении некоторых задач надежности и контроля: Дис. ... канд. техн. наук : 05.13.01 / Шелудько Олег Иванович – К., 1975. – 166 с

5. Павлов А. В., Степашко В.С. Рекуррентные алгоритмы расчёта коэффициентов и критериев селекции в релаксационном алгоритме МГУА // Кибернетика и вычислительная техника. – 2011. – Вып. 165. – С. 72-82.