Реализация облегчённой версии паттерна Модель/Представление/Контроллер в кроссплатформенном программном обеспечении для спектрофотометра Брюэра для продолжения долгосрочных измерений озона и ультрафиолетовой радиации на глобальной сети

Савиных В.В. (amita@ifaran.ru)
Институт физики атмосферы им. А.М. Обухова РАН, Москва, Россия

Спектрофотометр Брюэра MkII #043 ВНС ИФА РАН (Кисловодск, 2070 м)

Введение. Озон является одним из ключевых атмосферных газов, который участвует во многих фотохимических реакциях и вносит значительный вклад в климатическую систему Земли. Он сильно поглощает поступающее в стратосферу УФ излучение и излучает в тепловом ИК диапазоне. Наблюдаемое восстановление озонового слоя всё ещё может быть неустойчивым, и задача обеспечения однородности наблюдений за озоном по сравнению с измерениями предыдущих десятилетий остаётся актуальной. Одной из старейших глобальных систем, предоставляющих данные озона, является сеть автоматизированных спектрофотометров Брюэра (рис. 1), которая функционирует с начала 1980-х годов и включает около 80-ти наземных станций в 40-ка странах мира (рис. 2).

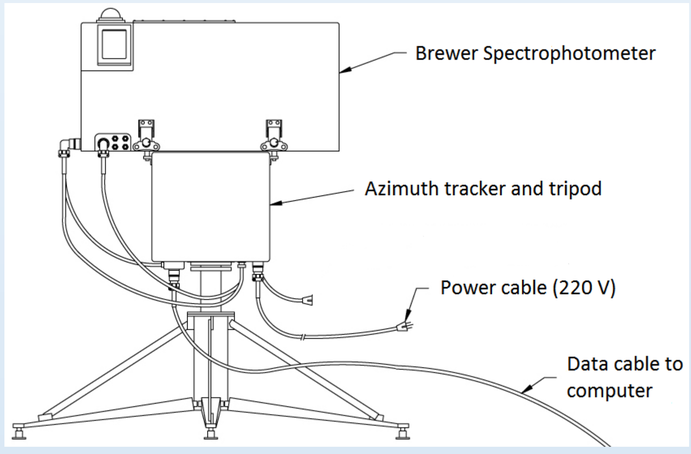

Рис. 1. Спектрофотометр Брюэра для измерений ОСО и УФ радиации (рисунок Kipp & Zonen).

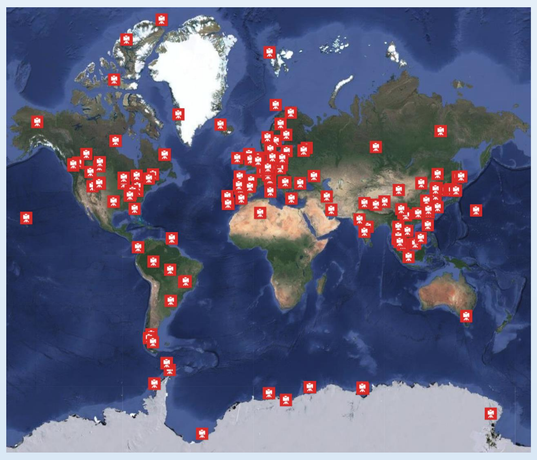

Рис. 2. Местоположения спектрофотометров Брюэра по миру (рисунок Kipp & Zonen).

Цели. Существующее ПО для управления Брюэром создавалось более 35 лет назад для ПК с ОС MS-DOS и нуждается в реконструкции для поддержания наблюдений на современных вычислительных платформах. Разрабатываемое новое кроссплатформенное ПО для Брюэра способно работать на компьютерах (Windows, Linux, macOS), смартфонах и планшетах (Android, iOS) с современными многозадачными ОС и одновременно имеет единую кодовую базу (рис. 3). Данное ПО реализует свой «облегчённый» вариант архитектурного паттерна проектирования *Модель/Представление/Контроллер* (МПК) и разделяет код приложения на три взаимодействующие части, при этом каждая из них может изменяться независимо (рис. 4).

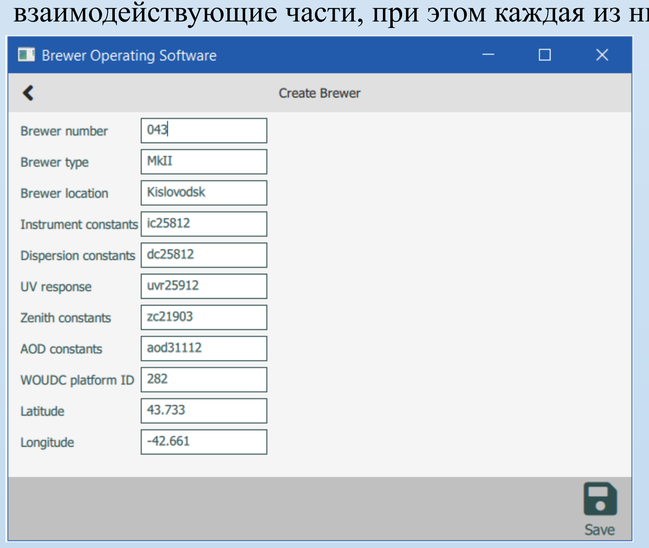

Рис. 3. Форма создания записи о новом Брюэре вместе с панелями навигации и команд.

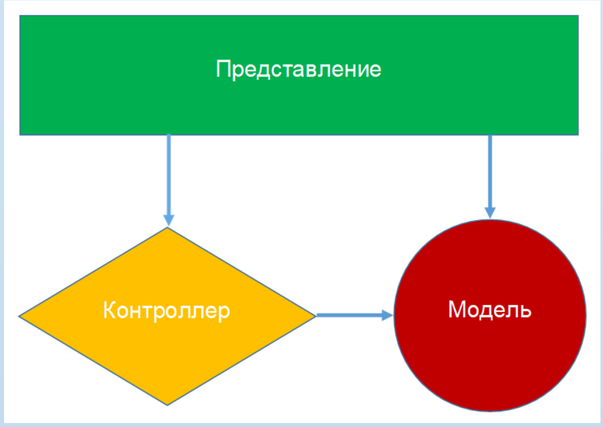

Рис. 4. Паттерн МПК с взаимодействиями между Моделью, Представлением и Контроллером.

Модель – динамическая структура для управления данными, логикой и правилами приложения. *Представление* отображает данные в интерфейсе пользователя или спектрофотометра, передаёт команды контроллеру для извлечения данных. *Контроллер* делегирует ответственность за данные модели. Представление зависит от контроллера и модели, т.к. отправляет команды контроллеру и отображает данные, содержащиеся в модели. Контроллер зависит от модели, т.к. делегирует ей работу, но не зависит от представления. Модель не зависит ни от контроллера, ни от представления. Паттерн МПК в новом ПО для Брюэра используется для отделения бизнес-логики приложения от интерфейсов пользователя и спектрофотометра.

Модели данных. Приложение Брюэра представляет данные в иерархической или реляционной форме, рационализированные в отдельные объекты. «Корневой» объект является родителем нескольких дочерних объектов и коллекций объектов. Каждый объект имеет собственный набор элементов данных, которые могут быть любого типа. Все измерения и инструментальные константы отображаются в классы моделей данных, они описывают спектрофотометры (`Brewer` – корневой (родительский) объект иерархии данных), местоположения (`Location` – дочерний объект), измерения (`Measurement` – коллекция дочерних объектов) и др. (рис. 5). Все модели наследуются от базового класса `Entity`, который представляет отношения «родитель/потомок»; он поддерживает три коллекции – карту декораторов данных (отдельные поля модели), карту сущностей (дочерние объекты) и карту коллекций сущностей (дочерние коллекции объектов) (рис. 6). При объявлении полей данных в классах моделей используются декораторы данных (напр., `IntDecirator`) вместо обычных типов данных C++ (`int`, и т.д.). Декораторы проверяют и преобразуют входные данные и предоставляют описательную метку рядом с текстовым полем в представлении, например, «Номер Брюера» (рис. 7, 8).

```
class Brewer : public Entity

private fields:
  IntDecorator *id
  StringDecorator *number
  Specification *specification
  EntityCollection<Measurement> *measurements
  ...

properties:
  IntDecorator *Id READ id
  StringDecorator *Number READ number
  Specification *Specification READ specification
  EntityCollection<Measurement> *Measurements READ measurement
  ...

access methods:
  IntDecorator *id()
  StringDecorator *number()
  Specification *specification()
  EntityCollection<Measurement> *measurements()
  ...
```

Рис. 5. Класс модели `Brewer` описывает спектрофотометр; ключевое слово `READ` определяет методы доступа для чтения данных.

```
class Entity : public QObject

private fields:
  map<QString, DataDecorator *> decorators
  map<QString, Entity *> entities
  map<QString, EntityCollection *> entityCollections

access methods:
  map<QString, DataDecorator *> &decorators()
  map<QString, Entity *> &entities()
  map<QString, EntityCollection *> &entityCollections()

protected methods:
  DataDecorator *addDataItem()
  Entity *addChildEntity()
  EntityCollection *addChildCollection()

signals:
  void dataDecoratorsChanged()
  void childEntitiesChanged()
  void childCollectionChanged()
```

Рис. 6. Класс `Entity` – базовый для всех моделей; предоставляет связь «родитель/потомок» между моделями данных.

```
class IntDecorator : public DataDecorator

private field:
  int value

property:
  int Value READ value WRITE setValue NOTIFY valueChanged

access methods:
  int value()
  IntDecorator &setValue(int value)
  void update(DataDecorator &decorator)

signal:
  void valueChanged()
```

Рис. 7. Класс `IntDecorator` – декоратор данных для целого типа; `NOTIFY` определяет событие `valueChanged()` изменения свойства `Value`.

```
class DataDecorator : public QObject

private field:
  QString label

property:
  QString Label READ label

access methods:
  QString &label()
  virtual void update(DataDecorator &decorator)
```

Рис. 8. Класс `DataDecorator` – базовый для всех декораторов данных: `label` описывает текстовую метку в представлении.

Контроллеры. Вместо жесткого кодирования данных в интерфейсах их можно динамически получать из контроллеров. Класс `MasterController` – центральный элемент паттерна МПК в разрабатываемом ПО – содержит вспомогательные контроллеры, модели данных и обычные данные (рис. 9). Класс `NavigationController` реализует навигацию между представлениями для перемещения по приложению. Панели команд представлений – контекстно-зависимые и содержат разные наборы кнопок, поведение последних реализует класс `Command` (рис. 11). Он включает иконку, описательный текст и метод `canExecute()` для определения доступна кнопка или нет. Представление имеет список команд, поставляемый интерфейсу пользователя через класс `CommandController` (рис. 10), который инкапсулирует списки команд, методы доступа к ним и методы, вызываемые при нажатии на кнопки для выполнения бизнес-логики ПО. Приложение использует встраиваемую реляционную БД SQLite в качестве постоянного хранилища. Класс `DatabaseController` содержит методы подключения к БД, создания её таблиц с полями согласно моделям данных, а также реализует базовые функции CRUD, т.е. создания, чтения, обновления и удаления данных (рис. 12).

```
class MasterController : public QObject

private fields:
  NavigationController *navigation
  CommandController *command
  DatabaseController *database
  Brewer *newBrewer
  ...

properties:
  NavigationController *Navigation READ navigation
  CommandController *Command READ command
  DatabaseController *Database READ database
  Brewer *NewBrewer READ newBrewer
  ...

access methods:
  NavigationController *navigation()
  CommandController *command()
  DatabaseController *database()
  Brewer *newBrewer()
  ...
```

Рис. 9. Класс `MasterController` инкапсулирует контроллеры, модели и обычные данные, динамически доставляя их в представления.

```
class CommandController : public QObject

private fields:
  Brewer *newBrewer
  QList<Command *> createBrewer
  QList<Command *> findBrewer
  ...

properties:
  QList<Command *> CreateBrewer READ createBrewer
  QList<Command *> FindBrewer READ findBrewer
  ...

access methods:
  QList<Command *> createBrewer()
  QList<Command *> findBrewer()
  ...

public slots:
  void onCreateBrewerSave()
  void onFindBrewerSearch()
  ...
```

Рис. 10. Класс `CommandController` включает списки команд представлений, методы доступа и выполнения логики.

```
class Command : public QObject

private fields:
  QString iconCharacter
  QString description
  std::function<bool()> canExecute

properties:
  QString IconCharacter READ iconCharacter
  QString Description READ description
  bool CanExecute READ canExecute NOTIFY canExecuteChanged

access methods:
  QString &iconCharacter()
  QString &description()
  bool canExecute()

signals:
  void canExecuteChanged()
  void executed()
```

Рис. 11. Класс `Command` описывает поведение кнопки в интерфейсе пользователя; метод `canExecute()` определяет доступность кнопки.

```
class DatabaseController : public QObject

private field:
  QSqlDatabase database

private methods:
  bool initialise()
  bool createTables()
  QString sqliteVersion()

crud methods:
  bool createRow()
  Entity *readRow()
  bool updateRow()
  bool deleteRow()
  EntityCollection *find()
```

Рис. 12. Класс `DatabaseController` содержит методы подключения к БД, создания в ней таблиц и функции CRUD.

Представления. Представления, в которые передаются данные – это класс спектрофотометра и интерфейс пользователя с навигационной платформой, реализованный на *QML* (Qt Modeling Language) – иерархическом декларативном языке разметки пользовательского интерфейса с синтаксисом, похожим на *JSON*. `MasterView` – корневое представление приложения, которое всегда присутствует на мониторе и содержит глобальную панель навигации и панель контента, где содержимое добавляется и удаляется по мере необходимости. Оно включает в себя дочерние представления (`CreateBrewerView`, `EditBrewerView` и т.д.) с наборами контекстно-зависимых команд для выполнения различных действий, например, для сохранения записи в БД (рис. 3). Панель навигации имеет пункты меню, перемещающие пользователя по приложению. Панель содержимого представляет собой стек дочерних представлений. Переход к различным областям приложения достигается путем замены дочернего представления на панели содержимого. Панель команд – необязательный элемент – предоставляет пользователю командные кнопки, которые снабжены иконками с кратким описанием внизу.

Заключение. На примере кроссплатформенного ПО для спектрофотометра Брюэра, разрабатываемого группой исследователей из ИФА РАН, показана реализация "облегченного" шаблона проектирования Модель/Представление/Контроллер, который служит для отделения бизнес-логики приложения от интерфейсов спектрофотометра и пользователя. Паттерн МПК реализован с помощью кроссплатформенного каркаса разработки приложений Qt на объектно-ориентированном ЯП C++14, в качестве постоянного хранилища используется встраиваемая приложение реляционная СУБД SQLite.

Литература:

1. *Ball W.T. et al.* Evidence for a continuous decline in lower stratospheric ozone offsetting ozone layer recovery // Atmos. Chem. Phys. 2018. Vol. 18, P. 1379-1394, doi: 10.5194/acp-18-1379-2018.
2. *Oram D.E. et al.* A growing threat to the ozone layer from short-lived anthropogenic chlorocarbons // Atmos. Chem. Phys. 2017. Vol. 17, P. 11929-11941, doi: 10.5194/acp-17-11929-2017.
3. *Savinykh V.V., Postylyakov O.V.* On development of cross-platform software to continue long-term observations with the Brewer Ozone Spectrophotometer // Proc. SPIE. 2018. V. 10786. 107860V. P. 1-12. doi: 10.1117/12.2515121.
4. *Gamma E., Helm R., Johnson R., Vlissides J.* Design Patterns: Elements of Reusable Object-Oriented Software, First Edition. Upper Saddle River, NJ: Addison-Wesley Professional, 1994. 395 p.
5. *Sherriff N.* Learn Qt 5. Birmingham: Packt Publishing, 2018. 346 p.

Работа выполнена в рамках темы ГЗ №0129-2019-0002.