**4.2. База пакетов (управляющая оболочка и интеллектуальный конвертор данных)**

Прикладные сервисы в составе системы реализуются в виде вычислительных пакетов, ориентированных на решение конкретных прикладных задач. Для унификации доступа к вычислительным пакетам и их абстрагирования от остальных компонентов платформы используется компонент − база пакетов (БП).

Абстрагирование вычислительных пакетов заключается в организации единообразного доступа, процедурам обмена данными с пакетами, а так же обработки входных и выходных данных.

Основные функции БП (с указанием ответственного модуля):

* хранение общей декларативной информации о пакете, о параметрах запуска (репозиторий);
* хранение процедур обработки данных (репозиторий);
* обеспечение возможности встраивания новых пакетов или новых версий пакетов (репозиторий);
* предоставление общей информации декларативной и императивной информации о пакете по запросу (библиотека). Предоставление информации о версии пакета, поставщике, дате выпуска, режимах распараллеливания, а также о входных и выходных параметрах.
* выполнение процедур работы над входными и выходными данными пакета (библиотека). Извлечение данных из файлов, сборка файлов на основе параметров, сборка командной строки запуска, переменных окружения и т.п.

В прототипе компонента база пакетов предназначена для встраивания только приложений, запускаемых в пакетном режиме. Компонент не поддерживает графические приложения, а также приложения, взаимодействующие с пользователем или другими агентами во время исполнения (это не относится к приложениям, распараллеливаемым средствами библиотек MPI, OpenMP и т. п.)

Программный компонент делится на два модуля: интерфейсная библиотека и репозиторий пакетов. Интерфейсом базы пакетов является программная библиотека: для использования данного компонента библиотека должна входить в состав клиентского модуля.

Репозиторий пакетов — это удаленное централизованное хранилище информации о вычислительных пакетах. С технической точки зрения — это web-сервер, который обеспечивает доступ к файловой структуре определенного вида. Разбиение компонента на два модуля было произведено для обеспечения высокой масштабируемости, гибкости и быстродействия такой архитектуры. На рисунке 4.2.1 представлена архитектура БП.

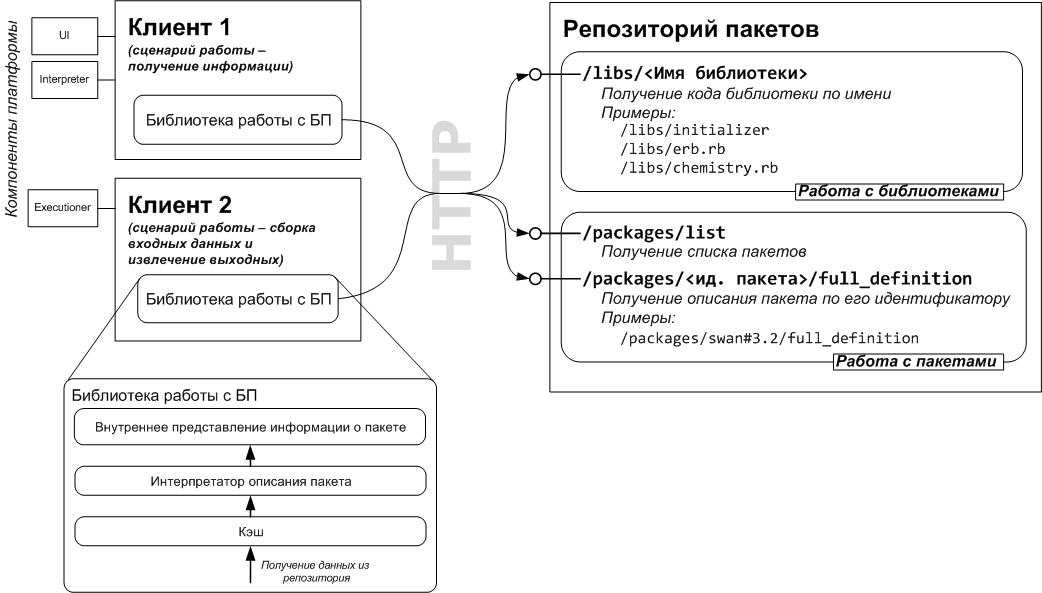


Рисунок 4.2.1 − Архитектура компонента БП

В основе БП лежит идея представления всей информации о пакете в виде скриптов (интерпретируемых программ). Она позволяет совместить в одном описании декларативную (общая информация) и императивную (процедуры работы с данными) составляющие. В качестве языка для написания скриптов был выбран интерпретируемый язык Ruby, а для удобства работы с файлами описания на основе Ruby был разработан прототип предметно-ориентированного (DSL) языка программирования EasyPackage. Такой подход позволил использовать мощный набор существующих инструментов обработки данных в Ruby для генерации входных файлов и анализа выходных (например, механизмы шаблонизации файлов). Для уменьшения количества запросов к репозиторию пакетов в библиотеке был реализован кэш, который во время работы создает локальный репозиторий пакетов и сохраняет там вновь загруженные скрипты и файлы. Кэширование при этом производится до интерпретации скриптов (см. рисунок 4.2.1).

Репозиторий пакетов имеет в своей основе следующую файловую структуру. Список пакетов, встроенных в платформу, хранится в файле "/packages/list". Все данные, связанные c пакетом, хранятся в "/packages/<идентификатор пакета>/", в том числе скрипт описания пакета ("full\_definition"), шаблоны входных фалов, логотипы и т.д. В качестве идентификатора пакета (PackageIdentifier) используется связка имени пакета и его версии (в строковом представлении). Доступ ко всем хранимым файлам осуществляется за счет выполнения HTTP GET-запроса на адрес, состоящий из базового адреса сервера (например, <http://localhost:8080/>) и относительного пути до файла (например, packages/list).

Работа с библиотекой БП основана на использовании клиентом внутреннего представления информации о пакете в виде объектов платформы CLR .NET (рис. 4.2.2).

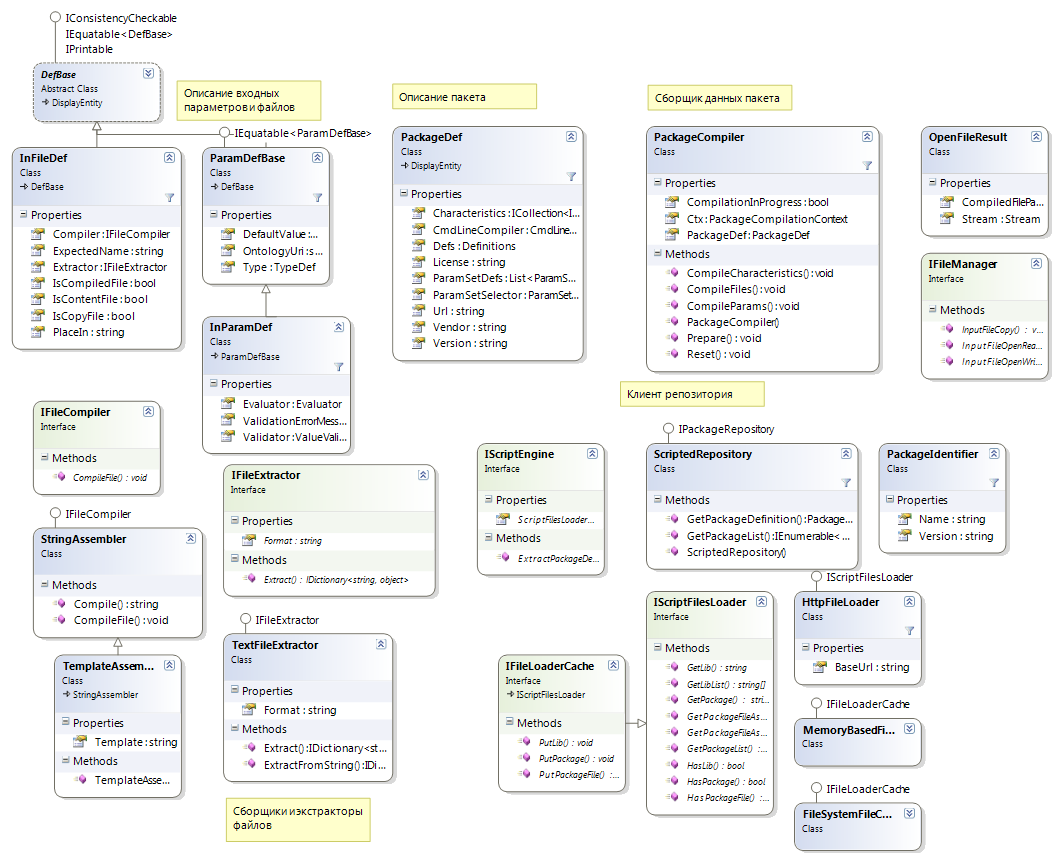


Рис. 4.2.2 − Диаграмма классов интерфейсной библиотеки БП

Пакет (вычислительный пакет) — это исполняемое приложение, запускаемое в пакетном режиме, которое на вход принимает определенный набор входных файлов, параметров командной строки, переменных окружения и других источников данных, предоставляемых операционной системой при запуске, на выход генерирует набор выходных файлов. Описание пакета представляется в виде объекта класса PackageDef. Он включает в себя набор полей, несущих общую информацию о пакете: имя, версия, лицензия, поставщик, адрес сайта, а также набор дескрипторов входных и выходных параметров и файлов пакета, а также дескриптор командной строки.

Параметр пакета — это элемент данных, имеющий имя, тип и значение. Параметр может быть входным или выходным. Тип параметра (TypeDef) может быть одним из следующих: строка (StringType), логический тип (BoolType), число с плавающей точкой (DoubleType), перечислимый тип (EnumType), целое число (IntType), список (ListType), дата и время (DateTimeType).

Дескриптор входного параметра (InParamDef) описывает входной параметр пакета. Помимо стандартных для параметров полей в описание входят значение по умолчанию, идентификатор в онтологии, режим запуска (TargetMode), валидатор (Validator), вычислитель (Evaluator). Значение по умолчанию проставляется, если пользователем данный параметр не был определен. Идентификатор в онтологии необходим только для параметров, которые отображаются на онтологическое описание пакета в компоненте хранения знаний.

Поле «режим запуска» определяет, в каком режиме используется данный параметр: низкоуровневом (Raw) или высокоуровневом (Meta). В низкоуровневом режиме параметр транслируется напрямую либо в параметр командной строки, либо в переменную окружения (его можно воспринимать как параметр запуска исполняемого приложения). Высокоуровневый параметр не определяет способа использования при запуске: он может быть включен в собираемый входной файл, может транслироваться в низкоуровневый параметр, может являться основой для вычислимых параметров.

Валидатор (Validator) — это процедура проверки значения параметра на то, что оно принадлежит области допустимых значений. Вычислитель (Evaluator) необходим для вычислимых параметров, он определяет, каким образом значение данного параметра будет вычислено на основе других параметров. Валидатор и вычислитель относятся к императивной части описания пакета.

Дескриптор входного файла (InFileDef) определяет, как будет обработан входной файл пакета. Дескриптор определяется местоположением файла после генерации (PlaceIn), ожидаемым именем (ExpectedName), экстрактором (Extractor), сборщиком (Compiler). Для файлов, как и для параметров, существует два режима: низкоуровневый и высокоуровневый.

Описание файла на низком уровне соответствует конкретному типу файлов, принимаемых пакетом (файл конфигурации, файл данных). На высоком уровне описанный файл может являться основой для параметров или файлов. Местоположение входного файла указывает, где (по какому относительному пути) файл должен находиться перед запуском. Ожидаемое имя файла указывается в случае, когда пакет ожидает наличия файла с конкретным именем. Экстрактор входного файла определяет процедуру извлечения данных из входного файла в параметры или другие файлы. Сборщик определяет, как собрать данный файл на основе набора параметров.

Дескрипторы выходного параметра (OutParamDef) и выходного файла (OutFileDef) эквивалентны входным дескрипторам с той разницей, что в описании выходного параметра отсутствует значение по умолчанию, а в описании выходного файла местоположение и ожидаемое имя говорят, где искать сгенерированный файл.

Для фильтрации входных файлов и параметров существует механизм наборов параметров (ParamSet), который позволяет указывать, какие параметры необходимы и опциональны для данного набора.

По типу использования информации, полученной от БП, можно выделить два сценария работы с базой пакетов: получение информации и работа с данными. В качестве источника информации базу пакетов использует компонент построения сценариев исполнения для выполнения проверок перед запуском шага:

* проверка наличия запускаемого пакета;
* проверка соответствия типов параметров запуска, указанных пользователем в скрипте WF, типам параметров, приведенных в спецификации пакета;
* проверка попадания указанных значений параметров в область допустимых значений.

Компонент диалогового интерфейса использует БП для:

* отображения общей информации о пакете;
* отображения списка доступных пакетов;
* интерактивной подсказки пользователю (IntelliSense);
* осуществления проверок, аналогичных проверкам в компоненте интерпретации WF.

Использование БП с целью получения, отображения или обработки информации о пакете производится посредством получения значений полей и вызовов методов объектов класса PackageDef.

В полной мере функциональность БП используется компонентом управления исполнением для:

* формирования параметров перед запуском пакета;
* вычисления значений характеристик расчета с целью передачи в компонент планирования исполнения и компонент моделирования производительности для оценки времени работы при выборе целевого ресурса;
* формирования входных файлов, командной строки и других элементов окружения перед запуском пакета на конкретном ресурсе;
* разбора выходных файлов пакета.

Входными данными для библиотеки БП являются: базовый URL репозитория, пользовательские параметры. Входными файлами репозитория пакетов являются хранимые в нем:

* скрипты описания пакетов;
* вспомогательные библиотеки на языке Ruby;
* шаблоны входных файлов;
* логотипы, текст лицензии и др. вспомогательные файлы общего характера.

Выходными данными библиотеки БП являются объекты .NET, структура которых представлена на рис. 4.2.2.

Выходными данными репозитория являются любые хранимые в нем файлы:

* скрипты описания пакетов;
* вспомогательные библиотеки на языке Ruby;
* шаблоны входных файлов;
* логотипы, текст лицензии и др. вспомогательные файлы общего характера.

Пример описания пакета Swan в виде Ruby-скрипта представлен в листинге 4.2.1. Вычислительный пакет SWAN является ядром прототипа сервиса волнового моделирования. SWAN это открытое программное обеспечение, распространяемое в рамках лицензии GNU. В основе SWAN лежит волновая модель третьего поколения, предназначенная для расчета случайных коротких ветровых волн в прибрежных морских районах и внутренних водах.

Для интеграции пакета SWAN необходимо описать его взаимодействие с прототипом платформы, опираясь на формат представления входных и выходных данных компонента БП. Платформенный скрипт описания пакета SWAN (см. листинг 4.2.1) позволяет определить уровень абстракции и интерпретации в работе с входными и выходными параметрами, что обеспечивает гибкость и упрощение процедуры взаимодействия пользователя с платформой на этапе запуска задания. Тем самым необходимость в определении на входе тех или иных параметров по умолчанию устраняется. Это приводит к более эффективному использованию пакета в рамках выполняемого сервиса.

Листинг 4.2.1. Основная часть скрипта описания параметров пакета SWAN

inputs {

meta param {

name "startCalcDate"

required

display\_as "Start Calculation Date"

type datetime

validator lambda { |val, ctx| val < Time.now }

}

meta param {

name "endCalcDate"

required

display\_as "End Calculation Date"

type datetime

depends\_on ["startCalcDate"]

evaluator lambda { |ctx| ctx.startCalcDate }

validator lambda { |val, ctx| val >= ctx.startCalcDate }

}

auto file {

name "inHirlam"

required

filename "hirlam.zip"

place "/data"

}

raw file {

name "inConfig"

required

filename "config.ini"

assembler erb\_template rtext("swanConfig.erb")

place "/"

}

cmdline lambda { |ctx| "{0}" } }

outputs {

auto file\_group {

name "swanouts"

filters ["[^/]\*\.zip$"] }

}

Как видно из листинга 4.2.1, все входные параметры описаны со следующими дополнительными атрибутами:

1. «необходимый» (required) – данный атрибут отражает условие присутствия этого параметра при запуске, в описании пакета SWAN это – startCalcDate**,** endCalcDate**,** inHirlam**,** inConfig**;**
2. «информация о параметре» (Display\_as) – данный атрибут отражает текстовую информацию о параметре в интерфейсе пользователя.
3. «валидатор»(validator) – необходим для проверки валидности значения параметра, например, endCalcDate должен быть больше либо равен startCalcDate;
4. «тип» (type) – необходим для отсеивания низкоуровневых ошибок, связанных с типизацией данных.

Помимо представленных параметров на примере приведенного выше скрипта определяются также и общие характеристики пакета SWAN.