МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Генеральный директор
ЗАО «АйТи»

_____ Бакиев О.Р.
"12" декабря 2011 г.

УТВЕРЖДАЮ
Ректор НИУ ИТМО

_____ Васильев В.Н.
"20" декабря 2011 г.

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ–БАЗА ПАКЕТОВ CLAVIRE/PACKAGEBASE

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.80066-06 13 35-ЛУ

Представители
Организации-разработчика

Руководитель разработки,
профессор НИУ ИТМО

_____ Бухановский А.В.
"19" декабря 2011 г.

Ответственный исполнитель,
с.н.с. НИУ ИТМО

_____ Луценко А.Е.
"19" декабря 2011 г.

Нормоконтролер
ведущий инженер НИУ ИТМО

_____ Позднякова Л.Г.
"19" декабря 2011 г.

2011

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН
RU.СНАБ.80066-06 13 35-ЛУ

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ – БАЗА ПАКЕТОВ CLAVIRE/PACKAGEBASE

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ. 80066-06 13 35

ЛИСТОВ 21

2011

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

АННОТАЦИЯ

Документ содержит описание программного компонента-базы пакетов CLAVIRE/PackageBase RU.СНАБ.80066-06 01 35, абстрагирующего остальные компоненты МИТП от вычислительных пакетов, встроенных в платформу в качестве прикладных сервисов и унифицирующий доступ к информации о них. Программный компонент – база пакетов разработан в ходе выполнения проекта «Создание распределенной вычислительной среды на базе облачной архитектуры для построения и эксплуатации высокопроизводительных композитных приложений» (Договор № 21057 от 15 июля 2010 г., шифр 2010-218-01-209) в рамках реализации постановления Правительства РФ № 218 «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства».

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Компонент-база пакетов CLAVIRE/PackageBase RU.СНАБ.80066-06 01 35 абстрагирует остальные компоненты платформы от вычислительных пакетов, встроенных в платформу в качестве прикладных сервисов, и унифицирует доступ к информации о них. Программный компонент состоит из двух модулей: удаленного репозитория пакетов и интерфейсной библиотеки, предназначенной для работы с репозиторием и структурами данных представления информации о пакете.

Программный модуль интерфейсной библиотеки написан на языке C# 4.0 и поставляется в виде .NET-сборки для платформ .NET 4.0 и Silverlight 4.0. Репозиторий пакетов организован как файловый сервер, работающий по протоколу http. В качестве реализации модуля пригоден любой существующий web-сервер, использующий файловую систему для хранения данных (Apache, Microsoft IIS, lighthttpd, nginx).

Программный модуль интерфейсной библиотеки предназначен для функционирования на аппаратных системах с архитектурой процессора: x86, x86_64, IA64. Модуль требует для своей работы наличия следующего системного программного обеспечения: ОС семейства Windows NT (версии старше Windows 2000), .NET 4.0.

Репозиторий пакетов предназначен для функционирования на аппаратных системах с любой архитектурой процессора, для которой доступна реализация web-сервера (в том числе x86, x86_64, IA64). Модуль требует для своей работы наличия следующего системного программного обеспечения: ОС, для которой доступна реализация web-сервера (в том числе семейств Windows NT, Linux, BSD); web-сервер, работающий по протоколу http и использующий файловую систему для хранения информации.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Программный компонент-база пакетов (БП) предназначен для обеспечения уровня абстракции между другими компонентами платформы и вычислительными пакетами, встроенными в систему. БП унифицирует доступ к информации о пакетах, а также к процедурам обмена данными с пакетами, обработки входных и выходных данных.

Основные функции БП (с указанием ответственного модуля):

1) хранение общей декларативной информации о пакете, о параметрах запуска (репозиторий);

2) хранение процедур обработки данных (репозиторий);
3) обеспечение возможности встраивания новых пакетов или новых версий пакетов (репозиторий);
4) предоставление общей 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;
5) выполнение процедур 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 текущей версии компонент предназначен для встраивания только приложений, запускаемых в пакетном режиме. Компонент не поддерживает графических приложений, а также приложений, взаимодействующих с пользователем или другими агентами во время исполнения (это не относится к приложениям, распараллеливаемым средствами библиотек MPI, OpenMP и т.п.).

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

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 приложения. Репозиторий пакетов – это удаленное централизованное хранилище информации о вычислительных пакетах, в котором для каждого пакета обязательно находится его описание и, возможно, дополнительные файлы, требуемые для его запуска и сборки входных файлов (см. ниже).

Разбиение компонента на два модуля было произведено для обеспечения масштабируемости, гибкости и быстродействия архитектуры. На рис. 3.1 представлена архитектура программного компонента.

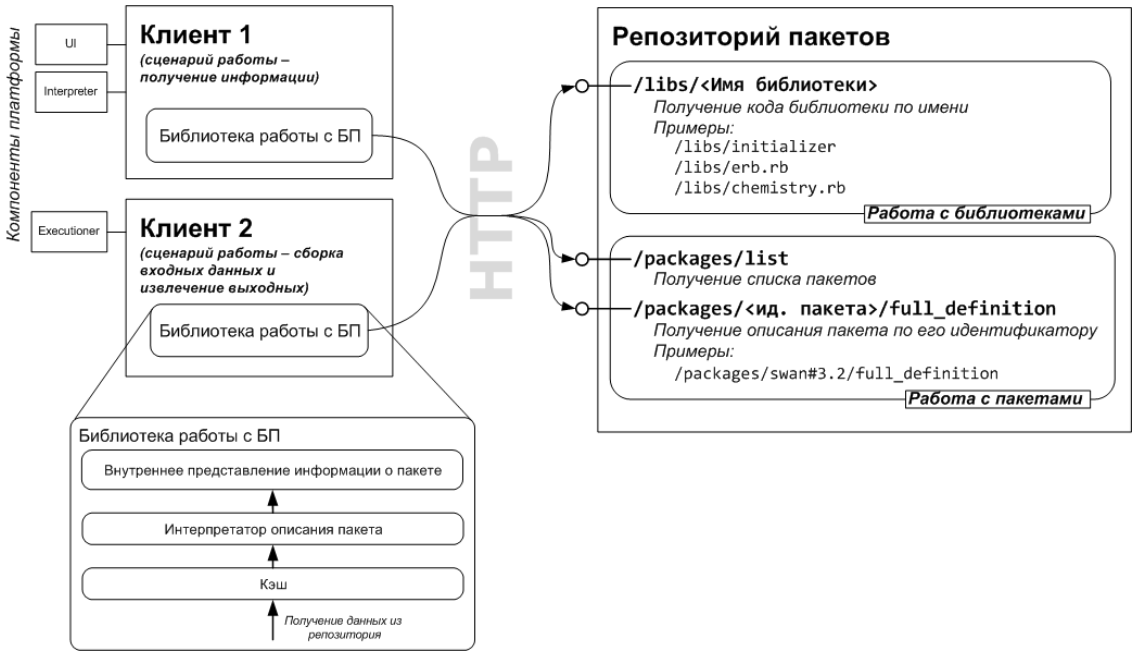

Рисунок 3.1 – Архитектура компонента

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 предметно-ориентированный (DSL) язык 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 и предоставляет лишь содержимое файлов.

3.1. Структура репозитория пакетов

С технической точки зрения репозиторий пакетов – это web-сервер, который обеспечивает доступ к файловой структуре определенного вида.

Репозиторий пакетов имеет в своей основе следующую файловую структуру. Список пакетов, встроенных в платформу, хранится в файле /packages/list в виде списка идентификаторов пакетов, разделенных переводом строки. Все данные, связанные с пакетом, хранятся в папке /packages/<идентификатор пакета>/ (где <идентификатор

пакета> – это уникальное для системы имя, обычно совпадающее с именем самого пакета), в том числе: скрипт описания пакета (файл 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).

Структура основных интерфейсов и классов репозитория пакетов приведена на рис. 3.2.

Рисунок 3.2 – Основные интерфейсы и классы репозитория пакетов

Основными для репозитория являются три интерфейса: IScripFileLoader, IScriptEngine и IPackageRepository. Интерфейс IPackageRepository является реализацией шаблона проектирования «фасад» для доступа к базе пакетов (см. описание ниже), а интерфейсы IScriptEngine и IScriptFileLoader представляют собой классы, ответственные соответственно за интерпретацию скриптов описания пакетов и загрузку файлов из определенного вида источника (например, http или файловая система).

Пример использования классов, реализующих приведенные интерфейсы, приведен в листинге 3.1.

Листинг 3.1

```
IScriptFilesLoader fileLoader = new FileSystemFileLoader(@"path\to\folder");
IScriptEngine scriptEngine = new RubyScriptEngine();
IPackageRepository repository = new ScriptedRepository(fileLoader, scriptEngine);
```

3.1.1. Интерфейс IScriptFilesLoader

IScriptFilesLoader – интерфейс для классов, способных работать с файловой структурой базы пакетов, описанной в разделе 3.1, безотносительно способа получения файлов.

Открытые методы

- string *GetPackage*(PackageIdentifier packageIdentifier) – возвращает скрипт описания пакета (full_definition, см. выше) по его идентификатору (*packageIdentifier*) в виде строки;
- string *GetLib*(string name) – возвращает файл из директории lib по имени (*name*);
- string *GetPackageFileAsText*(PackageIdentifier packageIdentifier, string textFilePath) – возвращает файл пакета по идентификатору пакета (*packageIdentifier*) и относительному пути к файлу (*textFilePath*) в виде строки;
- bool *HasLib*(string name) – проверяет, имеется ли в директории lib файл с именем *name*;
- bool *HasPackage*(PackageIdentifier packageIdentifier) – проверяет, имеется ли в базе пакетов описание пакета с идентификатором *packageIdentifier*;
- bool *HasPackageFile*(PackageIdentifier packageIdentifier, string textFilePath) – проверяет, имеется ли файл пакет с идентификатором *packageIdentifier* и относительным путем *textFilePath*;
- string[] *GetLibList*() – возвращает именя файлов из папки lib;
- IEnumerable<PackageIdentifier> *GetPackageList*() – возвращает список идентификаторов пакетов, доступных в базе паектов.

3.1.2. Интерфейс IScriptEngine

Интерфейс для классов, способных переводить описание скрипта из базы пакетов в объектное представление интерфейсной библиотеки базы пакетов.

Открытые методы

- ModeDef ExtractPackageDef(string initScript, string script) – извлекает определение пакета из переданного в текстовом виде скрипта (*script*) и скрипта инициализации (*initScript*);
- ScriptFilesLoaderWrapper ScriptFilesLoaderWrapper – возвращает обертку для интерфейс IScriptFilesLoader (см. выше).

3.1.3. Интерфейс IModeDefResolver

Интерфейс для классов, способных по имени пакета возвращать объектное представление его описания.

Открытые методы

- ModeDef *ResolveMode*(ModeQName modeName) – возвращает по имени пакета *modeName* его описание в объектном виде;
- bool *CanResolveMode*(ModeQName modeName) – проверяет, может ли быть разрешено имя пакета *modeName*.

3.1.4. Интерфейс IPackageRepository

Интерфейс для классов, представляющих собой фасад для работы с базой пакетов.

Открытые методы

- IEnumerable<PackageIdentifier> *GetPackageList*() – возвращает список идентификаторов пакетов, зарегистрированных в базе пакетов;
- ModeDef *GetPackageDefinition*(PackageIdentifier packageIdentifier) – возвращает объектное описание пакета по его идентификатору (*packageIdentifier*).

3.2. Структура интерфейсной библиотеки

Работа с библиотекой программного компонента основана на использовании клиентом внутреннего представления информации о пакете в виде объектов платформы .NET.

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 выходе генерирует набор выходных файлов. Описание пакета представляется в виде объекта класса ModeDef. Он включает в себя набор полей, несущих общую информацию о пакете: имя, версия, лицензия, поставщик, адрес сайта, 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).

Рисунок 3.3 – Диаграмма классов интерфейсной библиотеки БП

Для представления различных элементов описания пакетов была разработана иерархия интерфейсов и реализующих их классов, приведенная на рис. 3.3.

Описание этих интерфейсов приведено в пунктах ниже.

3.2.1. Интерфейс IDisplayEntity

Интерфейс для классов, содержащих поля, используемые для отображения пользователю.

Открытые свойства

- string *Name* – возвращает внутреннее имя элемента;
- string *DisplayName* – возвращает или устанавливает отображаемое имя элемента;
- string *Description* – возвращает или устанавливает текстовое описание элемента.

3.2.2. Интерфейс IDependent

Интерфейс классов, содержащих зависимости от других элементов.

Открытые свойства

- IEnumerable<Dependency> *Dependencies* – возвращает список зависимостей от других элементов.

3.2.3. Интерфейс IDef

Корневой интерфейс для элементов описания базы пакетов. Реализует интерфейсы IConsistencyCheckable, IDisplayEntity, IDependent.

Открытые свойства

- DefVisibility *Visibility* – возвращает или устанавливает видимость элемента;
- DynamicContextPredicate *Enabled* – возвращает или устанавливает предикат, определяющий, должен ли использоваться данный элемент при обработке описания пакета. Вычисляется динамически.

3.2.4. Интерфейс IInDef

Корневой интерфейс для описания входных параметров пакета. Реализует интерфейс IDef.

Открытые свойства

- DynamicContextPredicate *Required* – возвращает или устанавливает предикат, определяющий, требуется ли установка данного элемента от пользователя. Вычисляется динамически.

3.2.5. Интерфейс IOutDef

Корневой интерфейс для описания выходных параметров пакета. Реализует интерфейс IDef.

Открытые свойства

- DynamicContextPredicate *Expected* – возвращает или устанавливает предикат, определяющий, ожидается ли появление данного элемента после выполнения пакета при текущих условиях. Вычисляется динамически.

3.2.6. Интерфейс IParamDef

Корневой интерфейс для описания параметров пакета. Реализует интерфейс IDef.

Открытые свойства и методы

- string *ValidationErrorMessage* – устанавливает или возвращает сообщение об ошибке валидации значения параметра;

- TypeDef *Type* – устанавливает или возвращает тип параметра;
- ValueValidator *Validator* – устанавливает или возвращает предикат, определяющий, корректно ли значение параметра, переданного от пользователя. Вычисляется динамически;
- bool *HasValidator* – имеется ли у параметра предикат валидации;
- ObjectEvaluator *Evaluator* – устанавливает или возвращает делегат, вычисляющий значения параметра на основе других параметров. Вычисляется динамически;
- bool *HasEvaluator* – установлено ли у параметра свойство *Evaluator*;
- ValidationResult *Validate*(object value, DynamicContext ctx) – проверяет, допустимо ли значение параметра, переданное в *value*.

3.2.7. Интерфейс IFileDef

Корневой интерфейс для описания файлов пакета. Реализует интерфейс IDef.

Открытые свойства

- string Path – устанавливает или возвращает путь к файлу относительно рабочей директории пакета;
- IFileAssembler Assembler – устанавливает или возвращает сборщик файла (см. ниже);
- bool HasExtractor – установлено ли свойство *Extractor*;
- bool HasAssembler – установлено ли свойство *Assembler*;
- IFileExtractor Extractor – устанавливает или возвращает класс, производящий извлечение параметров из файлов. Вычисляется динамически;
- string ExpectedName – устанавливает или возвращает ожидаемое имя файла.
- bool IsPackage – принадлежит ли файл непосредственно к пакету или нет. В случае значения False – файл является внутренним файлом базы пакетов или пользовательским файлом, который не копируется при запуске пакета.

3.2.8. Классы описания пакета

Дескриптор входного параметра (InParamDef) описывает входной параметр пакета. Помимо стандартных для параметров полей в описание входят значение по умолчанию, 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 Visibility определяет видимость элемента пакета. Элемент может быть видимым и приватным. В первом случае при генерации интерфейса пользователя данный элемент будет отображен, во втором – нет. Это необходимо для разграничения параметров и файлов, которые требуется задать пользователю (или которые он может непосредственно получить после запуска пакета), от параметров и файлов, которые используются на промежуточных этапах работы базы пакетов.

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 (Path), ожидаемым именем (ExpectedName), экстрактором (Extractor), сборщиком (Assembler). Для файлов, как и для параметров, существует два режима видимости: публичный и приватный.

Файл также может принадлежать или не принадлежать пакету (свойство IsPackage). Значение IsPackage=true соответствует конкретному типу файлов, принимаемых пакетом (файл конфигурации, файл данных). При IsPackage=false 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, 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 (ModeDef), который позволяет указывать, какие параметры необходимы и опциональны для данного набора.

По типу использования информации, полученной от программного компонента, можно выделить два сценария работы с ним: получение информации и работа с данными.

В качестве источника информации базу пакетов использует компонент интерпретации WF CLAVIRE/FlowSystem RU.СНАБ 80066-06 01 20 перед запуском шага для выполнения проверки:

1) наличия запускаемого пакета;
2) соответствия типов параметров запуска, указанных пользователем в скрипте WF, типам параметров, приведенных в спецификации пакета;
3) попадания указанных значений параметров в область допустимых значений.

Компонент взаимодействия с пользователем CLAVIRE/Ginger RU.СНАБ 80066-06 01 21 использует БП для:

1) отображения общей информации о пакете;
2) отображения списка доступных пакетов;
3) интерактивной подсказки пользователю (Code completion);
4) осуществления проверок, аналогичных проверкам в компоненте интерпретации WF.

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

В полной мере функциональность БП используется компонентом исполнения WF CLAVIRE/Executor RU.СНАБ 80066-06 01 29 для:

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

Рассмотрим процесс работы компонента пакетного исполнения с БП (учитывая при этом алгоритм работы БП) во время запуска вычислительной задачи (подготовка, запуск, обработка результатов).

3.3. Порядок работы компонента

Запуск задачи на исполнение начинается после того, как компонент интерпретации WF передает полное описание задачи, куда входят параметры запуска, идентификатор пакета, идентификаторы входных файлов, компоненту исполнения WF.

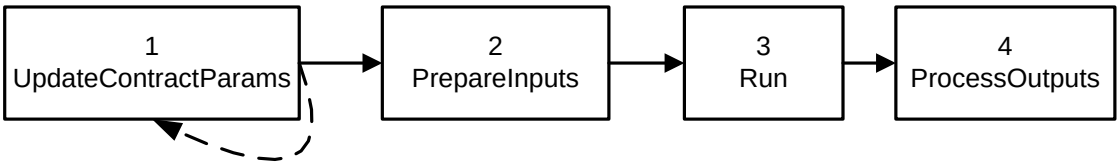

Рисунок 3.4 – Порядок работы базы пакетов

Первым этапом работы (этапы нумеруются согласно рис. 3.4) БП является инициализация входных параметров (UpdateContractParams), переданных пользователем. При инициализации параметров производится их проверка на попадание в область допустимых значений за счет вызова валидатора. При обработке входных параметров и файлов, как и на других этапах, учитываются зависимости параметров и файлов друг от друга таким образом, что конкретный файл или параметр вычисляется только после того, как вычислены все элементы, от которых он зависит. Этот этап может повторяться несколько раз по мере ввода пользователем новых параметров и файлов. При этом ранее вычисленные параметры и файлы сохраняются в контексте двигателя базы пакетов (PackageEngine).

Второй этап – подготовка входных файлов (PrepareInputs). Подготовка производится на основе контекста, который уже сформирован к этому моменту (в контекст входят параметры, переданные пользователем, и параметры, извлеченные из входных фалов). Также на этом этапе вычисляются характеристики. Характеристика – это специальный вид параметра, который является общим для группы пакетов и вычисляется на основе переданных и вычисленных параметров. Основная задача характеристик расчета – использование в компонентах планирования исполнения WF и моделирования производительности. Далее все значения параметров, входящих в контекст, передаются в компонент планирования исполнения WF, где формируются параметры запуска задачи (число ядер, режим распараллеливания). Параметры запуска возвращаются в компонент планирования исполнения. Далее происходит собственно формирование входных файлов: здесь для каждого из входных файлов (которые не были переданы пользователем) вызывается соответствующий сборщик, который на базе имеющихся параметров и файлов собирает входной файл.

Когда задача сформирована, происходит запуск (этап 3: Run) на исполнение на конкретном ресурсе.

После этого, на четвертом шаге (ProcessOutputs), из выходных файлов извлекаются выходные параметры. Затем на основе полученных параметров высчитываются вычислимые выходные параметры, формируются выходные файлы (за счет экстракторов и сборщиков), результат возвращается в интерпретатор, а далее – пользователю.

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Интерфейсная библиотека реализована на основе платформы Microsoft .NET версии 4.0 и Silverlight версии 4.0. Языком разработки является C# версии 4.0. Данный модуль поставляется в виде .NET-сборки (библиотека dll) и предназначен для использования в рамках платформы .NET (в ее стандартной реализации для ОС Windows или для реализации Mono для ОС Linux/UNIX) или Silverlight (в ее стандартной реализации для ОС Windows/Mac или для реализации Moonlight для ОС Linux/UNIX). Требования к аппаратному обеспечению, на котором будет устанавливаться программный модуль:

- архитектура процессора – x86, x86_64, IA64;
- минимальный объем оперативной памяти – 500 МБ;
- минимальный объем свободного пространства на жестком диске – 10 ГБ;
- минимальная тактовая частота процессора – 1 ГГц.

Репозиторий пакетов предназначен для работы на машине и ОС, для которых существует http-сервер, позволяющий организовать отдачу файлов из файловой системы. Тип файловой системы не важен. Клиентам компонента должен быть обеспечен доступ к репозиторию по сети (протокол http). Требования к аппаратному обеспечению, на котором будет устанавливаться программный модуль:

- архитектура процессора – x86, x86_64, IA64;
- минимальный объем оперативной памяти – 2 ГБ;
- минимальный объем свободного пространства на жестком диске – 100 ГБ;
- минимальная тактовая частота процессора – 2 ГГц.

5. ВЫЗОВ И ЗАГРУЗКА

Для вызова репозитория пакетов необходимо запустить службу выбранного web-сервера. Перед запуском сервера должна быть обеспечена следующая файловая структура: создана директория для хранения библиотек (libs), в библиотеки добавлен инициализирующий скрипт (libs/initializer), создана директория для хранения описания пакетов (packages), создан файл со списком пакетов (packages/list).

Интерфейсная библиотека поставляется в виде отдельной сборки, поэтому до начала работы она должна быть подключена к проекту .NET-приложения. Сценарий работы с интерфейсной библиотекой включает несколько шагов: создание объекта

пакетного репозитория (при инициализации репозитория необходимо указать базовый URL), получение описания пакета (ModeDef), формирование контекста работы с БП (входные параметры), создание объекта сборщика (ModeCompiler) и двигателя базы пакетов (PackageEngine), выполнение остальных операций по формированию и обработке данных (см. рис. 3.4). Для получения информации о пакете достаточно выполнить первые три действия и работать с PackageDef. В листинге 5.1 приведен фрагмент работы с библиотекой.

Листинг 5.1. Фрагмент программы сборки входных данных для пакета Testp с использованием библиотеки БП

```
IScriptFilesLoader fileLoader = new FileSystemFileLoader(@"D:\usr\pb");
IScriptEngine scriptEngine = new RubyScriptEngine();
IPackageRepository repository = new ScriptedRepository(fileLoader, scriptEngine);

compiledModeDef = null;
ModeCompiler compiler = new ModeCompiler();
var result = compiler.Compile(new ModeQName("testp"), repository, out compiledModeDef)
;

if (!result.Succeeded)
    throw new Exception();

engine = new PackageEngine(compiledModeDef);

engine.UpdateContractParams(new Dictionary<string, string>());

IInputFileManager inputFileManager = new InputFileManager();
engine.PrepareInputs(inputFileManager);

// Here the package is being run

IOutputFileManager outputFileManager = new OutputFileManger();
engine.ProcessOutputs(new List<string> { "a", "b" }, outputFileManager);
```

6. ВХОДНЫЕ ДАННЫЕ

Входными данными для библиотеки БП являются: базовый URL репозитория, пользовательские параметры (рис. 6.1).

Входными файлами репозитория пакетов являются хранимые в нем:

- скрипты описания пакетов;
- вспомогательные библиотеки на языке Ruby;
- шаблоны входных файлов;
- логотипы, текст лицензии и другие вспомогательные файлы общего характера;
- входной файл list, который включает список встроенных пакетов, содержит набор строк, каждая строка представляет идентификатор пакета в формате <имя>#<версия> (рис. 6.2).

```
private static InterpreterContext GetGamessContext()
{
  return new InterpreterContext
  {
    Inputs = new PackageInputs
    {
      Params = new Dictionary<string, object>
      {
        {":basis", "6-31G"},
        {"method","HF+EOM-CC"},
        {":task","optimization"},
        {":functional","f0"},
        {":method","HF"},
        {"swap",true},
        {"hssend",false},
        {"hess",true},
      },
      Files = new Dictionary<string, ExternalFileDefinition>()
      {
        {
          "molecule_xyz",
          new ExternalFileDefinition{Locator="123",FileName="a.txt"}
        }
      }
    },
    RunMode = PackageRunMode.Meta
  };
}
```

Рисунок 6.1 – Фрагмент программы создания входного контекста для библиотеки БП

```
orca
gamess
testp
testp#2
```

Рисунок 6.2 – Пример файла list, содержащего список встроенных пакетов

Пример описания пакета NAMD в виде Ruby-скрипта представлен на рис. 6.3.

```
name "namd"

inputs {
      file {
            name "namd_config"
            required
            depends ["periodicBCMode", "out_freq", "binout_on_off", "num_of_iter", "timestep",
"stepspercycle", "nonbondedFreq", "fullElectFrequency", "pairlistdist",
                        "cutoff", "rigidBond", "useSettle", "rigidIterations"]
            path "/"
            filename "namd_config.conf"
            assembler erb_template rtext("namd_config_tamplate.erb")
      }

      public file{
            name "structure"
            required
            path "/src/"
            filename "structure.psf"
      }

      public file{
            name "coordinates"
            required
```

```
            path "/src/"
            filename "coordinates.pdb"
            extractor extractor CellBasisExtractor.new
      }

      public file {
            name "parameters"
            required
            path "/src/"
            filename "parameters.inp"
...
}

outputs{

      public file{
        name "logfile"
        display "Logfile"
            filename "log.log"
            path "/"
    }

      public file{
        name "coor"
        display "Coordinate file"
            filename "result.coor"
            path "/output/"
    }

      ...
}

prepare_package
```

Рисунок 6.3 – Пример фрагмента описания пакета NAMD

7. ВЫХОДНЫЕ ДАННЫЕ

Выходными данными библиотеки БП являются объекты .NET, структура которых представлена на рис. 3.3.

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.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

DSL	Domain-specific language
SOAP	Simple Object Access Protocol (простой протокол доступа к объектам)
WCF	Windows Communication Foundation (программный фреймворк)
WF	Workflow (поток заданий)
URL	Uniform Resource Locator (Единый указатель ресурсов)
БП	компонент – база пакетов
КИ	Компонент интерпретации WF
КПИ	компонент пакетного исполнения
МИТП	Многопрофильная инструментально-технологическая платформа

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	изменённых	заменённых	новых	аннулированных					