МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Генеральный директор
ЗАО «АйТи»
_____ Бакиев О.Р.
"29" декабря 2011 г.

УТВЕРЖДАЮ
Ректор НИУ ИТМО
_____ Васильев В.Н.
"29" декабря 2011 г.

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

ПРОГРАММНЫЙ КОМПОНЕНТ ИНТЕРПРЕТАЦИИ WF CLAVIRE/FLOWSYSTEM

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.80066-06 13 20-ЛУ

Представители
Организации-разработчика

Руководитель разработки,
профессор НИУ ИТМО
_____ Бухановский А.В.
"29" декабря 2011 г.

Ответственный исполнитель,
с.н.с. НИУ ИТМО
_____ Луценко А.Е.
"29" декабря 2011 г.

Нормоконтролер
ведущий инженер НИУ ИТМО
_____ Позднякова Л.Г.
"29" 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 Ошибка! Источник ссылки не найден.**0-ЛУ**

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

ПРОГРАММНЫЙ КОМПОНЕНТ ИНТЕРПРЕТАЦИИ WF CLAVIRE/FLOWSYSTEM

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ.80066-06 13 ОШИБКА! ИСТОЧНИК ССЫЛКИ НЕ НАЙДЕН.**0**

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

ЛИСТОВ 27

2011

АННОТАЦИЯ

Документ содержит описание программного компонента интерпретации WF CLAVIRE/FlowSystem RU.СНАБ.80066-06 01 20, предназначенного для исполнения композитных приложений в среде CLAVIRE, сформированных в виде скриптов на предметно-ориентированном языке EasyFlow (используемом в рамках МИТП для описания workflow). Программный компонент интерпретации WF 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 интерпретации (КИ) workflow CLAVIRE/FlowSystem RU.СНАБ.80066-06 01 20 предназначен для обработки описания композитных приложений, заданного в виде скрипта EasyFlow, обеспечения высокоуровневого представления структуры композитных приложений в форме цепочек заданий (workflow) и исполнения приложений в среде CLAVIRE. Данный компонент разработан на языке C# в виде web-сервиса с применением технологии WCF.

Компонент функционирует на аппаратных системах с архитектурой процессора x86, x86_64 и IA64. Для его работы необходимо следующее системное программное обеспечение: ОС семейства Windows NT (версии старше Windows 2000), .NET 4.0, Microsoft Internet Information Services (с версией старше 6.0).

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Компонент интерпретации WF отвечает за обработку композитных приложений и их исполнение в платформе CLAVIRE. Тем самым данный компонент обеспечивает высокоуровневое представление структуры композитных приложений и скрывает сложность и специфику работы с композитными приложениями от других компонентов подсистемы исполнения. Функциональное назначение КИ включает:

1) прием запросов на запуск WF, состоящих из описания WF на языке EasyFlow, контекста данных и контекста исполнения;
2) интерпретацию описания композитного приложения, полученного от компонента разбора скрипта EasyFlow RU.СНАБ.80066-06 01 19 в виде объектного представления;
3) исполнение WF с использованием компонента исполнения WF RU.СНАБ.80066-06 01 29 для запуска отдельных задач;
4) предоставление информации о структуре WF, составе и связи задач компоненту исполнения WF RU.СНАБ.80066-06 01 29 для осуществления планирования на уровне WF;
5) предоставление информации о WF компоненту мониторинга RU.СНАБ.80066-06 01 24.

Исполнение WF помимо запуска отдельных задач включает следующие возможности.

1) Интерпретация управляющих структур EasyFlow и динамическое формирование WF во время исполнения (например, возможность варьирования параметров).
2) Интерпретация императивного кода, связанного с постобработкой результатов исполнения пакетов.
3) Поддержка заданий длительного исполнения (LRWF).

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1. Принципы функционирования

КИ можно логически разбить на две части: программная библиотека интерпретации WF (Interpreter.dll) и WCF-сервис (FlowSystemService.svc), который предоставляет доступ к библиотеке извне.

Сервис компонента интерпретации WF является интерфейсом для всей подсистемы исполнения в CLAVIRE и разработан таким образом, чтобы обрабатывать множество запросов на исполнение композитных приложений. Интерфейсной операцией сервиса по запуску WF является PushJob, в которую клиент сервиса передает описание WF на языке EasyFlow, контекст данных (идентификаторы входных файлов для запуска задачи в хранилище данных, предоставленные пользователем) и контекст исполнения (например, ограничение на время жизни задачи). Переданный запрос помещается в очередь ожидания исполнения. На данном этапе клиенту возвращается идентификатор задачи и возвращается управление, т.е. метод PushJob можно назвать асинхронным. Очередь поступивших запросов обслуживается программными потоками обработки запросов, которые организованы в пул. Когда поток освобождается от выполняемой задачи, он берет новую задачу из очереди (если задач нет, то блокируется). Количество потоков в пуле задается в конфигурации сервиса. Схематически работа сервиса КИ представлена на рис. 3.1.

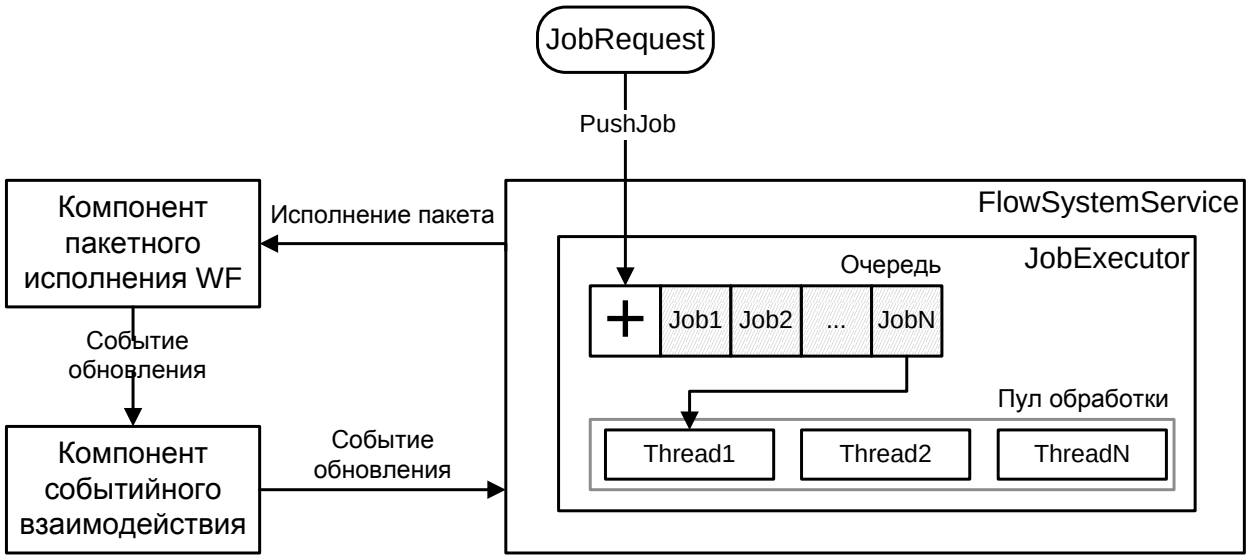

Рисунок 3.1 – Схема функционирования КИ WF

Процесс исполнения задачи потоком обработки включает в себя несколько последовательных этапов.

1) Разбор скрипта EasyFlow с помощью компонента разбора RU.СНАБ.80066-06 01 19. Проверка на отсутствие синтаксических и семантических ошибок.
2) Разбор контекста исполнения и контекста данных. Проверка корректности и целостности контекстов.
3) Первичное исполнение, в ходе которого выявляются структура WF, состав и конфигурация, входящих в него узлов. Полученная информация передается в компонент исполнения WF CLAVIRE/Executor, откуда она поступает в компонент планирования исполнения WF RU.СНАБ.80066-06 01 28.
4) Исполнение WF. Заключается в построении графа WF, исполнении отдельных узлов, обработке связей между узлами, учете порядка запуска.

Для исполнения WF сервис вызывает библиотеку интерпретации WF, обеспечив для нее инфраструктуру исполнения WF. В инфраструктуру входят сервисные механизмы, обеспечивающие взаимодействие КИ с компонентами платформы CLAVIRE. В частности, к инфраструктуре относятся:

1) механизм запуска отдельных пакетов (интерфейс IStepStarter);
2) механизм регистрации файлов (интерфейс IFileRegistry);
3) механизм управления задачами длительного исполнения (интерфейс LongRunningController);
4) файловое хранилище (интерфейс IStorage).

Конфигурирование библиотеки интерпретации производится за счет указания конкретных механизмов взаимодействия, которые реализуют перечисленные выше интерфейсы.

Первичное исполнение – это функция КИ, которая позволяет определить структуру WF до его реального запуска через компонент исполнения WF RU.СНАБ.80066-06 01 29. Данная функция необходима для обеспечения планирования на уровне связанных задач компонентом планирования исполнения WF RU.СНАБ.80066-06 01 28. Сложность получения информации о составе узлов в WF, их связях и параметрах состоит в том, что в описании WF такая информация содержится неявно, и для ее получения нужна интерпретация. Обусловлено это тем, что в описании WF присутствуют управляющие конструкции и динамические узлы, такие как, например, узел варьирования параметров (*sweep*). Получить значения всех входных параметров перед запуском пакета также не всегда представляется возможным из-за наличия между узлами связей по данным. Первичное исполнение представляет собой «фиктивную» интерпретацию WF, где реальные механизмы работы с инфраструктурой CLAVIRE заменены на заглушки, которые эмулируют нормальную работу компонентов платформы, используемых во время исполнения WF. Так, используемый механизм запуска пакетов вместо реального запуска сразу возвращает успешный пустой результат работы пакета. При этом интерпретация WF продолжается так, как если бы пакет реально отработал. Когда при интерпретации узла, который зависит от «фиктивно» запущенного пакета, становятся необходимыми значения реальных результатов работы этого пакета, например значения его выходных параметров, то в качестве их используется неопределенное значение (UndefinedValue). В результате первичного исполнения WF формируется частично определенный граф WF, где для узлов запуска пакета вместо реального значения входного параметра может фигурировать неопределенное. Полученная информация передается в компонент исполнения WF RU.СНАБ.80066-06 01 29 откуда она поступает в компонент планирования исполнения WF RU.СНАБ.80066-06 01 28 для создания плана исполнения узлов WF. После осуществления первичного исполнения начинается нормальное исполнение WF.

Помимо предоставления интерфейса для запуска WF сервис КИ предоставляет интерфейс для получения информации о исполняемых WF, а также интерфейс для управления WF во время его исполнения. Компонент мониторинга RU.СНАБ.80066-06 01 24 постоянно опрашивает сервис КИ, тем самым актуализируя информацию о состоянии WF. Из него данная информация поступает в компонент взаимодействия с пользователем RU.СНАБ.80066-06 01 21, а также другим клиентским модулям. Под управлением WF

подразумевается прием и выполнение команд над узлами WF или WF в целом, поступающих от внешних клиентов. Примерами таких команд являются приостановка, возобновление и остановка исполнения WF или конкретного узла. Набор поддерживаемых команд расширен для режима длительного исполнения (LRWF). Механизм управления подробнее рассмотрен ниже.

Таким образом, КИ представляет для внешних компонентов простой интерфейс запуска, тем самым скрывая сложность внутренней реализации и работы композитных приложений. Для оповещения клиентов об изменении состояния запущенных WF используется модель событийного взаимодействия, предоставляемая компонентом событийного взаимодействия RU.СНАБ.80066-06 01 23. Сервис также является клиентом компонента событийного взаимодействия и осуществляет функцию транслирования событий, относящихся к исполняемым WF, во внутреннюю шину событий. Событийная модель функционирования рассмотрена ниже.

Основным классом **программной библиотеки** интерпретации является DeclarativeInterpreter. Декларативный интерпретатор представляет собой класс, ответственный за управление исполнением WF в целом, т.е. за формирование последовательности выполнения тех или иных блоков WF, для которых выполняются все условия, необходимые для запуска (зависимости по данным и по управлению). Декларативный интерпретатор предназначен для исполнения WF на абстрактной инфраструктуре и должен быть сконфигурирован механизмами взаимодействия с инфраструктурой CLAVIRE (см. выше). Представим алгоритм исполнения WF в декларативном интерпретаторе.

1) Создание памяти интерпретатора, где хранятся значения всех переменных.
2) Извлечение параметров исполнения из описания WF. Параметры исполнения задаются в виде атрибутов с ключевым префиксом *flow* (например, [flow:Priority = @urgent]).
3) Создание узлов WF из внутреннего объектного представления, переданного компонентом разбора EasyFlow RU.СНАБ.80066-06 01 19. На данном этапе создаются и инициализируются узлы WF.
4) Установка зависимостей между узлами.
5) Проверка корректности полученного графа WF.
6) Создание фиктивных узлов WF: истока и стока. Установка зависимостей: для всех узлов, которые не зависели ни от одного узла WF, устанавливается зависимость от истока; для узла стока устанавливается зависимость от всех узлов, от которых не

зависел ни один узел. Это внутренний механизм, который упрощает схему запуска WF и его завершения.

7) Перенос контекста данных, переданных вместе со скриптом EasyFlow клиентом КИ, в память интерпретатора в виде глобальных переменных.
8) Генерация события о старте WF.
9) Установка всем узлам начального состояния (*state_initialized*).
10) Запуск цикла обработки событий.

В качестве механизма работы КИ используется **событийная модель**, а весь процесс интерпретации представлен циклом обработки событий. При этом внутреннее графовое представление WF является конечным автоматом, управляемым событиями. Все узлы WF являются конечными автоматами с определенным набором состояний (фиксированным для типа узла). Конечный автомат, управляемый событиями, может быть описан как множество отношений вида:

$$s \times e \to (a, s_1) : s, s_1 \in S, a \in A, e \in E, \quad (3.1)$$

где S – множество состояний, E – множество событий, A – множество действий. Если автомат находится в состоянии s и происходит событие e, то выполнятся действие a, и автомат переходит в состояние s_1.

В событийной модели обычно выделяют три типа объектов: событие, обработчик события, диспетчер событий. В случае КИ диспетчером событий является внутренняя событийная шина, которая позволяет оповещать о произошедшем событии все подписанные обработчики. Обработчиком события является узел WF (наследник класса *NodeBase*) или WF в целом. Обработка событий для узлов представлена в виде выполнения определенных действий из списка поддерживаемых. Для примера, все узлы поддерживают действие начала работы – *action_start.* Подписка на событие выражается в виде добавления условия на происхождение определенного события к выполняемому над узлом действию. Для примера, если узел B зависит от узла A по данным, то для действия *action_start* узла B будет зарегистрировано условие – срабатывание события узла A о завершении работы – *block_finished.*

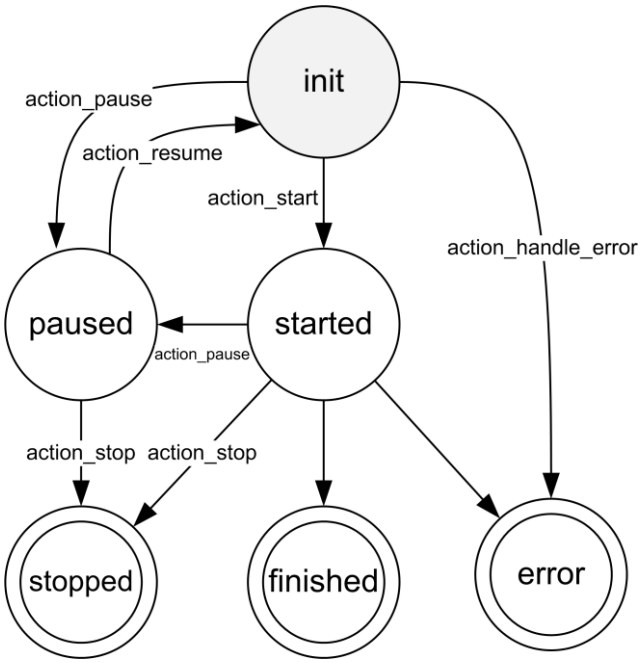

Рисунок 3.2 – Базовый граф состояний и переходов для узла *NodeBase*

На рис. 3.2 представлен граф состояний, который соответствует базовому типу узла – *NodeBase*. Все остальные типы узлов наследуются от указанного класса и расширяют представленный набор состояний и поддерживаемых действий. Все узлы начинают работу из начального состояния *init* (на рисунке и в тексте далее в именах состояний для простоты опущен префикс «*state_*», в коде компонента при обозначении состояний данный префикс присутствует). Если все события, указанные как условия для срабатывания действия *action_start* удовлетворены, то управление передается в код обработки действия узла, после чего узел переходит в состояние *started*. Переход между состояниями может осуществляться и без возникновения отдельного события (на рисунке такие переходы обозначены стрелками без подписей). При выполнении одного действия узел может последовательно перейти сначала в одно состояние, а затем в другое. Пусть, для примера, при начале выполнения действия *action_start* узел переходит в состояние *started*, после чего одна из внутренних проверок, заложенная в обработчик *action_start*, говорит о наличии ошибки. В таком случае узел сразу перейдет в состояние *error*.

Обработка ошибок в графе WF реализована на базе событийной модели: каждый узел поддерживает действие *action_handle_error*, которое инициируется, если в одном из узлов, являющимся родительским, возникает ошибка (переходит в состояние error). При этом узел сам обрабатывает данную ситуацию и определяет, переходить в состояние ошибки или нет. По умолчанию, при возникновении ошибки в графе WF, ошибка по дереву WF передается до стока, после чего весь WF приобретает состояние ошибки и останавливается.

На рис. 3.2 помимо рассмотренных состояний присутствуют два, касающиеся возможности **управления WF**: пауза (*paused*), остановка (*stopped*). Данные состояния реализованы для возможности внешнего воздействия на WF со стороны клиента КИ. В состояние паузы и остановки, как и в состояние ошибки, узел может перейти из любого другого, поэтому далее для простоты эти состояния на схемах опущены (рис. 3.3).

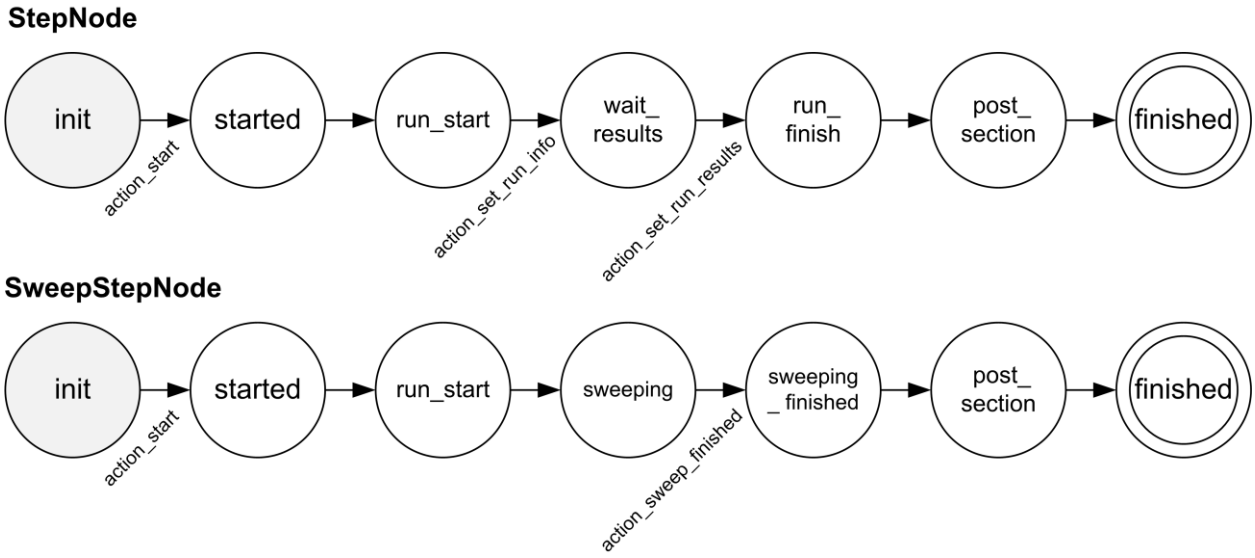

Рисунок 3.3 – Граф состояний и переходов для типов узлов:
StepNode (вверху), *SweepStepNode* (внизу)

Узел типа *StepNode* – основной блок исполнения WF, он представляет запуск отдельного пакета. При начале исполнения узел типа *StepNode* производит интерпретацию входных параметров, заданных в виде выражений EasyFlow. Для этого используется императивный интерпретатор EasyFlow (*ImperativeInterpreter*). Для выполнения кода на языке EasyFlow императивный интерпретатор содержит всю необходимую инфраструктуру: средства для хранения и обработки переменных и областей видимости, средства выполнения операторов и функций и др. **Память интерпретатора** для хранения переменных организована в виде дерева, узлами которого являются области видимости данных (*DataScope*). Поиск переменных выполняется в направлении вверх по дереву от области видимости текущего узла (экземпляра *BlockDataScope*) до корня (экземпляра *GlobalDataScope*), далее – вниз через области видимости отдельных узлов.

После интерпретации входных параметров *StepNode* формирует запуск пакета и выполняет его через соответствующий механизм (*IStepStarter*). Когда пакет, соответствующий узлу, запущен на инфраструктуре CLAVIRE (об этом сигнализирует полученное от компонента событийного взаимодействия оповещение о соответствующем

событии), *StepNode* получает от компонента исполнения WF RU.СНАБ.80066-06 01 29 информацию о ресурсе, на котором пакет запущен, и переходит в состояние *wait_results* – состояние ожидания результатов исполнения (см. рис. 3.3). После завершения исполнения пакета компонент исполнения WF RU.СНАБ.80066-06 01 29 генерирует соответствующее событие. Далее выполняется действие *action_set_run_results*, в задачи которого входит распаковка и проверка результатов, а также осуществление постобработки. **Постобработка** данных на EasyFlow позволяет обрабатывать результаты, полученные от запуска отдельной задачи путем интерпретации и исполнения императивного кода. В качестве языка для написания такого кода обработки поддерживается Ruby (IronRuby). Данная возможность позволяет гибко связывать узлы WF между собой, если по тем или иным причинам связь не удается установить путем описания пакета на EasyPackage. Постобработка данных позволяет также вычислять производные параметры из выходных параметров узла. Для узлов с варьированием параметров (*sweep*) данная функциональность необходима для агрегации полученных от нескольких шагов результатов. Интеграция языков EasyFlow и IronRuby производится за счет выполнения процедуры конвертации всего дерева значений, хранящегося в памяти интерпретатора EasyFlow, перед постобработкой к формату хранения памяти интерпретатора IronRuby (после обработки – обратно).

Узел типа *SweepStepNode* является узлом варьирования параметров. Он позволяет динамически во время исполнения WF создавать определенное число узлов *StepNode*, соответствующее списку значений для варьирования параметра. При указании одновременно нескольких варьируемых параметров используется один из двух механизмов варьирования: декартово произведение списков варьирования или строгое соответствие. Для примера, если для узла указано, что два его параметра должны варьироваться со списками значений a = [1,2] и b = [3,4], то в случае декартова произведения будет сгенерировано четыре узла с конфигурациями: a = 1, b = 3; a = 1, b = 4; a = 2, b = 3; a = 2, b = 4. А в случае соответствия получится два узла: a = 1, b = 3; a = 2, b = 4. Выбор механизма осуществляется за счет указания для узла атрибута *SweepMode*: *decart* – для декартова произведения (по умолчанию), *zip* – для соответствия.

3.2. Программная архитектура

Интерфейсом КИ является WCF-сервис FlowSystemService.svc, который позволяет остальным компонентам платформы обращаться к КИ. Данный сервис является оберткой для объекта класса *JobExecutor*, который повторяет интерфейс сервиса и отвечает за

предоставление всей функциональности КИ. Данный класс спроектирован согласно шаблону «одиночка» (*singleton*) и поддерживает одновременный доступ из нескольких потоков (*threadsafe*). Внутри объекта *JobExecutor* содержится три очереди для задач: необработанные, исполняемые, выполненные. В первую очередь задача помещается при поступлении от клиентов системы, во вторую – перемещается при начале обработки. Третья очередь используется для хранения задач, информацию о которых не удалось поместить в компонент хранения профилей исполнения WF RU.СНАБ.80066-06 01 32 (исключительная ситуация), чтобы не допустить потери данных. Все три очереди используются для получения информации об активных WF для последующей передачи в компонент мониторинга RU.СНАБ.80066-06 01 24.

Задача представлена внутри сервиса в виде объекта класса *Job*. Объекты этого класса являются активными, это означает, что они ответственны за собственное исполнение. Каждое задание при этом выполняется в отдельном потоке пула обработки, что позволяет реализовать параллельный режим обработки нескольких WF одновременно. Объект класса *Job* в основном используется для выполнения сервисных функций (многопоточный доступ), а также для подготовки WF к запуску.

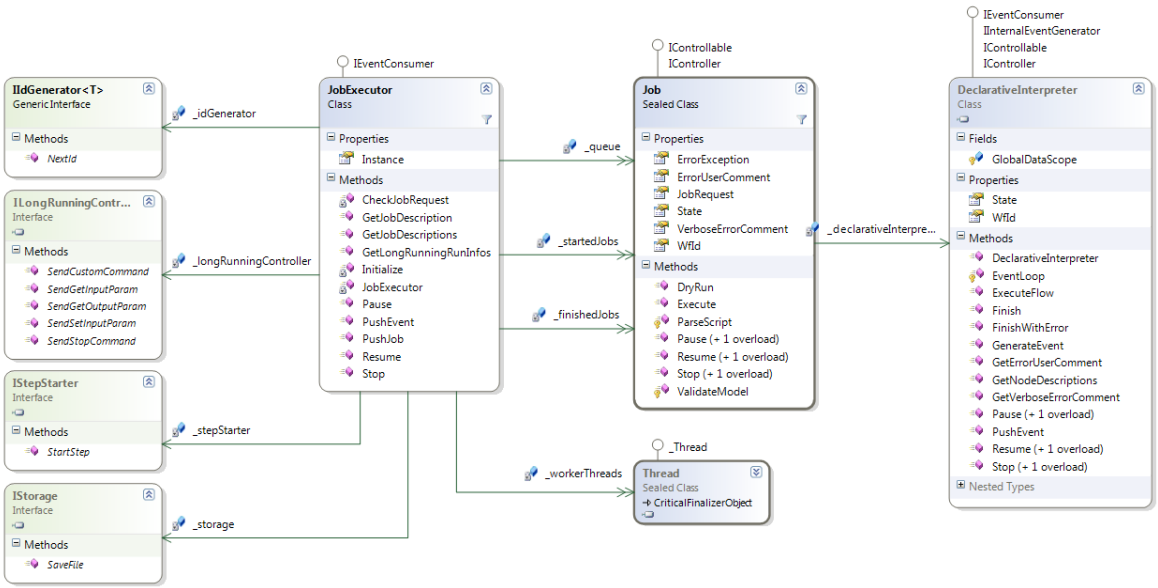

Рисунок 3.4 – Диаграмма основных классов сервиса КИ

Основная работа по исполнению WF выполняется в объекте класса *DeclarativeInterpreter*. Как было указано выше, обработка WF построена на событийной модели, поэтому основная задача *DeclarativeInterpreter* состоит в подготовке инфраструктуры для механизма обмена событиями и запуска цикла обработки событий.

Далее работа происходит внутри объектов – наследников класса *NodeBase*. Необходимо отметить, что использование событийной модели дало следующие преимущества.

1) Весь значимый код, который выполняет специфические для узлов WF действия, вынесен в не зависящие друг от друга классы узлов, объекты которых могут взаимодействовать унифицированным способом. Тем самым уменьшается связанность кода.
2) Набор типов узлов легко расширятся за счет добавления новых классов-наследников от *NodeBase*. При этом можно расширять набор событий и поддерживаемых действий (рис. 3.5).
3) Разделена логика процесса выполнения WF и управления этим процессом.
4) Так как код слабо связан, он хорошо ложится на модель параллельных и распределенных вычислений.

На рис. 3.5 представлено дерево наследования классов, соответствующих типам узлов и их контекстов. Контекст узла представляет собой конфигурацию узла.

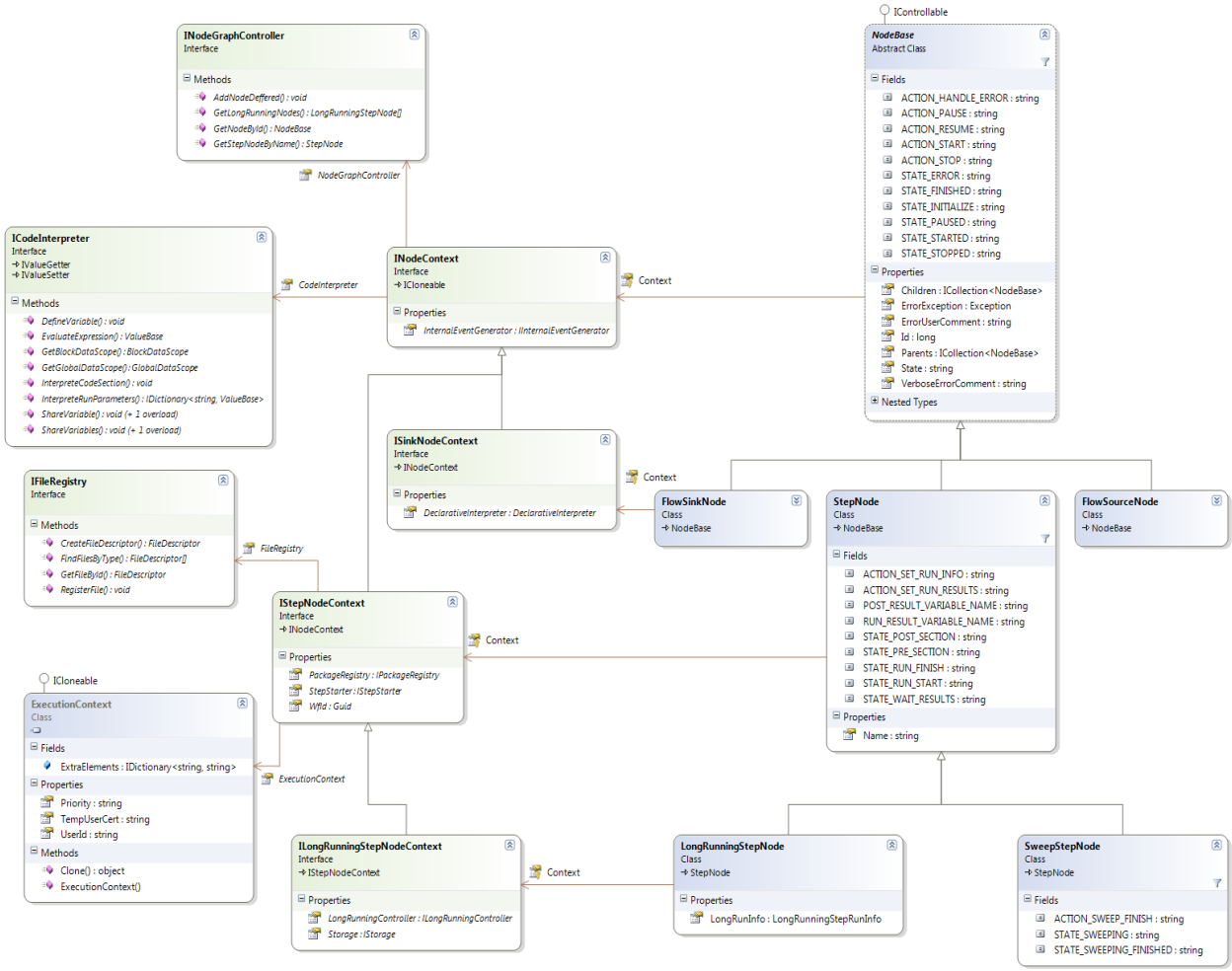

Рисунок 3.5 – Диаграмма классов узлов графа WF
и соответствующих им контекстов

3.3. Основные классы

Ниже приводятся сокращенные описания структуры и методов основных классов компонента интерпретации WF.

3.3.1. Интерфейс IFlowSystemService

Интерфейс сервиса КИ. Реализуется в классе FlowSystemService.

Методы

- Control – управление WF или конкретным узлом WF
    a. Входной параметр action (string) – команда (например, pause, stop, resume);
    b. Входной параметр wfId (Guid) – идентификатор WF;
    c. Входной параметр blockId (long) – идентификатор узла;
    d. Возвращаемое значение отсутствует.
- GetJobDescription – получение информации о WF
    a. Входной параметр wfId (Guid) – идентификатор WF;
    b. Возвращает информацию о WF (JobDescription).
- GetJobDescriptions – получение информации об исполняющихся WF
    a. Возвращает информацию об активных WF(JobDescription []).
- GetLongRunningTasks – получение информации о WF длительного исполнения с фильтрацией по имени пакета
    a. Входной параметр packageName (тип:string) – имя пакета для фильтрации;
    b. Возвращает массив LongRunningTaskInfo.
- PushJob – отправка команды на исполнение WF
    a. Входной параметр jobRequest (тип: JobRequest) – дескриптор задания;
    b. Возвращает идентификатор запущенного WF (Guid).

3.3.2. Класс JobExecutor

Основной класс сервиса. Содержит логику многопоточной обработки задач. FlowSystemService является оберткой для данного класса. Реализован по шаблону проектирования «одиночка». Поддерживает многопоточный доступ.

Методы

- WorkerFunc – функция потока обработки запросов
    a. Возвращаемое значение отсутствует.

- CheckJobRequest – базовая проверка корректности запроса на исполнение задания. Производится перед присваиванием ID и формированием объекта Job
    a. Входной параметр jobRequest (тип: JobRequest);
    b. Возвращает, прошел ли JobRequest проверку (тип: bool).
- GetJobDescription – получение полной информации о состоянии конкретного WF
    a. Входной параметр wfId (тип: Guid) – идентификатор WF;
    b. Информация о WF (тип: JobDescription).
- GetJobDescriptions – получение полной информации о состоянии всех активных WF
    a. Возвращает: Список контейнеров с данными о WF (JobDescription []).
- GetLongRunningRunInfos – получение информации о задачах длительного исполнения
    a. Возвращает информацию о задачах длительного исполнения (Dictionary<Guid,List<LongRunningStepRunInfo>>).
- Initialize – конфигурация сервиса, создание потоков обработки, привязка механизмов работы с CLAVRE
    a. Возвращаемое значение отсутствует.
- Pause – команда паузы. Маршрутизируется к WF или конкретному узлу
    a. Входной параметр wfId (тип: Guid) – идентификатор WF;
    b. Входной параметр blockId (тип: long) – идентификатор узла;
    c. Возвращаемое значение отсутствует.
- PushEvent – регистрация внешнего события. Событие маршрутизируется к WF
    a. Входной параметр flowEvent (тип: FlowEvent) – событие;
    b. Возвращаемое значение отсутствует.
- PushJob – постановка задачи в очередь на обработку. Проверка корректности jobRequest. Присвоение заданию идентификатора
    a. Входной параметр jobRequest (тип: JobRequest) – задание;
    b. Возвращает идентификатор задания (Guid).
- Resume – команда возобновления работы. Маршрутизируется к WF или к конкретному узлу
    a. Входной параметр wfId (тип: Guid) – идентификатор WF;
    b. Входной параметр blockId (тип: long) – идентификатор узла;
    c. Возвращаемое значение отсутствует.
- Stop – команда остановки. Маршрутизируется к WF или к конкретному узлу
    a. Входной параметр wfId (тип: Guid) – идентификатор WF;
    b. Входной параметр blockId (тип: long) – идентификатор узла;

c. Возвращаемое значение отсутствует.

Данные класса

- _queue (тип: SyncronizedBlockingQueue<Job>) – очередь поступающих задач. Поддерживает многопоточный доступ.

3.3.3. Класс Job

Задание на исполнение WF. Содержит информацию о WF. Несет функциональность по подготовке к запуску, исполнению WF. Использует компонент разбора скрипта EasyFlow. Поддерживает многопоточный доступ.

Методы

- Job – конструктор
    a. Входной параметр jobRequest (тип: JobRequest) – запрос выполнения задачи;
    b. Входной параметр wfId (тип: Guid) – идентификатор WF.
- DryRun – первичное исполнение WF. Использует DeclarativeInterpreter с фиктивными механизмами IStepStarter, IStorage, ILongRunningController
    a. Возвращаемое значение отсутствует.
- Execute – исполнение задания. Включает все этапы: разбор скрипта, подготовка, исполнение. Организован в виде цикла обработки конечного автомата (перехода между состояниями)
    a. Входной параметр flowSystemContext (тип: FlowSystemContext);
    b. Входной параметр stepStarter (тип: IStepStarter);
    c. Входной параметр storage (тип: IStorage);
    d. Входной параметр longRunningController (тип: ILongRunningController);
    e. Возвращаемое значение отсутствует.
- ExecuteJob – процедура исполнения задания. Производит первичный запуск, затем обычный. Блокирует поток обработки задания
    a. Возвращаемое значение отсутствует.
- ParseScript – разбор скрипта. Формирует: _parseResult
    a. Возвращаемое значение отсутствует.
- ValidateModel – разбор контекстов исполнения и данных. Проверка корректности контекстов. Формирует: _flowDataContext, _flowExecutionProperties
    a. Возвращаемое значение отсутствует.

Свойства

- ErrorException (тип: Exception) – исключение, вызвавшее ошибку;

- ErrorUserComment (тип: string) – описание ошибки для пользователя;
- JobRequest (тип: JobRequest) – структура, переданная клиентом;
- State (тип: JobState) – возвращает состояние задачи;
- VerboseErrorComment (тип: string) – подробный комментарий к ошибке;
- wfId (тип: Guid) – идентификатор WF.

3.3.4. Класс DeclarativeInterpreter

Интерпретатор декларативного кода. Отвечает за исполнение одного WF на абстрактной инфраструктуре. Механизм работы построен согласно событийной модели. Поддерживает многопоточный доступ.

Методы

- EventLoop – цикл обработки событий. В цикле извлекается новое событие из очереди и передается в FlowGraph через вызов Notify
    a. Возвращаемое значение отсутствует.
- CreateFlowGraph – создание FlowGraph из Flow (объектное представление WF, полученное от компонента разбора EasyFlow). Создает _flowGraph
    a. Возвращаемое значение отсутствует.
- EventLoop – цикл обработки событий. В цикле извлекается новое событие из очереди и передается в FlowGraph через вызов Notify
    a. Возвращаемое значение отсутствует.
- ExecuteFlow – исполнение WF. Основной метод, содержит всю логику подготовки и исполнения WF. Синхронный метод: поток блокируется до завершения WF
    a. Входной параметр wfId (тип: Guid) – идентификатор WF;
    b. Входной параметр flow (тип: Flow) – внутреннее представление скрипта WF;
    c. Входной параметр flowDataContext (тип: FlowDataContext) – контекст данных пользователя;
    d. Входной параметр flowExecutionProperties (тип: ExecutionContext) – опции исполнения WF;
    e. Возвращаемое значение отсутствует.
- ExtractExecutionParametersFromFlow – извлечение параметров исполнения из атрибутов, указанных в скрипте EasyFlow для WF,
    a. Входной параметр flow (тип: Flow) – объектное представление разобранного описания WF на EasyFlow;

b. Возвращает извлеченные параметры исполнения (IDictionary<string, string>).

- Finish – завершение работы интерпретатора. Вызывается из FlowSinkNodeClass
    a. Возвращаемое значение отсутствует.
- FinishWithError – остановка интерпретатора с ошибкой. Вызывается из FlowSinkNode
    a. Возвращаемое значение отсутствует.
- GenerateEvent – генерация нового события. Внутренний метод (для узлов, наследников NodeBase)
    a. Входной параметр eventName (тип: string) – название события;
    b. Входной параметр sourceBlockId (тип: long) – идентификатор узла;
    c. Возвращаемое значение отсутствует.
- Pause – приостановка работы интерпретатора (внешний метод)
    a. Возвращаемое значение отсутствует.
- PushEvent – добавление нового события в очередь событийной шины
    a. Входной параметр flowEvent (тип: FlowEvent) – событие;
    b. Возвращаемое значение отсутствует.
- Resume – возобновление работы интерпретатора. Внешний метод
    a. Возвращаемое значение отсутствует.
- SetupDataContext – регистрирация входных данных в памяти интерпретатора как глобальных переменных
    a. Возвращаемое значение отсутствует.
- Stop – остановка работы интерпретатора. Внешний метод
    a. Возвращаемое значение отсутствует.

Данные класса

- GlobalDataScope (тип: GlobalDataScope) – глобальная область видимости. Память интерпретатора.

Свойства

- wfId (тип: Guid) – идентификатор WF.

3.3.5. Класс FlowGraph

Графовая структура WF, построенная по зависимостям между блоками. Необходима для анализа циклов, выполнимости, поиска истоков и стоков. Также используется для передачи событий узлам.

Методы:

- void CheckNewNodes – проверка наличия новых узлов, созданных во время работы интерпретатора
    a. Возвращаемое значение отсутствует.
- AddNodeDeffered – отложенное добавление узлов для вызова из узлов. Новые узлы добавляются при вызове notify
    a. Входной параметр nodeBase (тип: NodeBase) – узел;
    b. Возвращаемое значение отсутствует.
- AddNodes – синхронное добавление узлов
    a. Входной параметр nodes (тип: IEnumerable< NodeBase >) – список узлов для добавления;
    b. Возвращаемое значение отсутствует.
- CheckNewNodes – проверка наличия новых узлов, созданных во время работы интерпретатора
    a. Возвращаемое значение отсутствует.
- GetErrorDescription – получение информации об ошибке за счет агрегации информации со всех узлов
    a. Возвращает строковое представление информации об ошибке.
- GetRealDependencies – метод получения реальной структуры WF из запусков пакетов, без динамических узлов
    a. Возвращает список зависимотей (List<Pair<long, long> >).
- GetStepNodeByName – поиск узла по имени
    a. Входной параметр name (тип: string) – имя узла;
    b. Возвращает узел (тип: StepNode).
- InitializeNodes – инициализация всех узлов в графе
    a. Возвращаемое значение отсутствует.
- InitSinkSource – поиск стоков и истоков и связь их с SinkNode и SourceNode. Вызывается после создания истока и стока
    a. Входной параметр flowSinkNode типа FlowSinkNode – сток;
    b. Входной параметр flowSourceNode типа FlowSourceNode – исток;
    c. Возвращаемое значение отсутствует.
- Notify – передача события для проверки подписок всем узлам. Предварительно проверяет наличие новых подписок
    a. Входной параметр flowEvent (тип: FlowEvent) – событие;
    b. Возвращаемое значение отсутствует.

- SetupDependencies – установка зависимостей между блоками, необходима для анализа структуры графа. На данном этапе созданы ВСЕ узлы (Nodes), кроме истокового и стокового
    a. Возвращаемое значение отсутствует.
- Validate – проверка корректности графа WF (на цикличность, на выполнимость)
    a. Возвращаемое значение отсутствует.

3.3.6. Класс NodeBase

Базовый класс для всех узлов. Содержит функциональность по подписке на события и механизм исполнения действий.

Открытые атрибуты

- STATE_INITIALIZE (тип: string) = "state_init";
- STATE_STARTED (тип: string) = "state_started";
- STATE_PAUSED (тип: string) = "state_paused";
- STATE_STOPPED (тип: string) = "state_stopped";
- STATE_FINISHED (тип: string) = "state_finished";
- STATE_ERROR (тип: string) = "state_error";
- ACTION_START (тип: string) = "action_start";
- ACTION_PAUSE (тип: string) = "action_pause";
- ACTION_RESUME (тип: string) = "action_resume";
- ACTION_STOP (тип: string) = "action_stop";
- ACTION_HANDLE_ERROR (тип: string) = "action_handle_error".

Свойства

- Id (тип:long) – идентификатор узла;
- Parents (тип:ICollection< NodeBase >) – родительские узлы, от которых зависит данный узел;
- Children (тип:ICollection< NodeBase >) – зависимые узлы;
- Context (тип:INodeContext) – контекст;
- State (тип:string) – состояние узла.

Методы

- AddDependencyTrigger – добавление зависимости между узлами
    a. Входной параметр from (тип: NodeBase) – от какого узла зависит;
    b. Входной параметр action (тип: string) – действие;

c. Входной параметр eventstring – событие;
d. Входной параметр triggerEventExpressionType – логическая операция для условий подписки;
e. Возвращаемое значение отсутствует.

- AssimilateEventToActions – усваивание информации о произошедшем событии, корректировка подписок
    a. Входной параметр flowEvent – событие;
    b. Входной параметр actions – список поддерживаемых действий;
    c. Возвращаемое значение отсутствует.
- void GenerateEvent – генерация события
    a. Входной параметр eventName – наименование события;
    b. Возвращаемое значение отсутствует.
- Subscribe – Подписка на событие и вызов метода. Этот метод можно вызывать внутри Node. Он устанавливает новую подписку в очередь подписок
    a. Входной параметр actionName (тип: string);
    b. Входной параметр flowEvent (тип: FlowEvent);
    c. Входной параметр triggerEventExpressionType – логическая операция для условий подписки;
    d. Возвращаемое значение отсутствует.

Данные класса

- _block (тип: BlockBase) – объектное представление узла, разобранное из скрипта EasyFlow.

3.3.7. Класс StepNode

Узел запуска пакета. Основной блок WF. Использует механизм IStepStarter для запуска пакета. Производит обработку результатов и постобработку.

Открытые атрибуты

- STATE_PRE_SECTION (string) = "state_pre_section";
- STATE_RUN_START (string) = "state_run_start";
- STATE_WAIT_RESULTS (string) = "state_wait_results";
- STATE_RUN_FINISH(string) = "state_run_finish";
- STATE_POST_SECTION (string) = "state_post_section";
- ACTION_SET_RUN_RESULTS (string) = "action_set_run_results";

- ACTION_SET_RUN_INFO(string) = "action_set_run_info";
- RUN_RESULT_VARIABLE_NAME (string) = "Result";
- POST_RESULT_VARIABLE_NAME (string) = "Post".

Свойства

- Name (тип: string) – имя узла;
- Context (тип: IStepNodeContext) – контекст.

Методы

- Action – обработка действия
    a. Входной параметр actionName (тип: string) – действие;
    b. Входной параметр arg (тип: object) – аргумент;
    c. Возвращаемое значение отсутствует.
- CloneNode – клонирование узла. Используется для динамического создания узлов (например, для варьирования параметров)
    a. Входной параметр newId (тип: long) – новый идентификатор;
    b. Входной параметр newName (тип: newName) – новое имя узла;
    c. Входной параметр newParams (тип: IEnumerable< NamedParameter >) – новые входные параметры;
    d. Возвращает клонированный узел (тип: NodeBase).
- GetStepRunParams – заполнение информации о параметрах для запуска пакета
    a. Входной параметр stepRunDescriptor типа StepRunDescriptor – описание запуска пакета;
    b. Входной параметр paramValues типа IDictionary< string, ValueBase > – значения параметров;
    c. Возвращает сформированные параметры (StepRunParameters).
- PostSection – постобработка
    a. Возвращаемое значение отсутствует.
- RunFinish – действие, завершение исполнения узла
    a. Входной параметр Result (тип: StepRunResult) – результат исполнения пакета;
    b. Возвращаемое значение отсутствует.
- RunStart – действие, запуск пакета на исполнение. Подготовка и запуск через механизм запуска
    a. Возвращаемое значение отсутствует.

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Все блоки КИ реализованы на основе платформы Microsoft .NET версии 4.0. Языком разработки является C#. Сам КИ реализован в виде сервиса WCF (FlowSystemService.svc).

КИ предназначен для использования в рамках платформы .NET (в ее стандартной реализации для ОС Windows или для реализации Mono для ОС Linux/UNIX). Для хостинга предполагается использование сервера IIS версии 7.0 и выше.

Требования к аппаратному обеспечению, на котором будет устанавливаться КИ: процессор x86 или совместимый, объем оперативной памяти не менее 1 ГБ, объем свободного места на жестком диске не менее 500 МБ.

5. ВЫЗОВ И ЗАГРУЗКА

КИ реализован в виде SOAP WCF-сервиса платформы .NET. Для запуска сервиса используется стандартный механизм используемого web-сервера. Загрузка сервиса в этом случае выполняется web-сервером автоматически по мере поступления запросов от клиентов сервиса.

КИ предоставляет интерфейс в виде WCF-сервиса, реализованного классом FlowSystemService (см. спецификацию IFlowSystemService в разделе 3.3.1). Вызов сервиса производится стандартным для технологии WCF способом: по опубликованному описанию сервиса (WSDL) необходимо создать прокси-класс, через который осуществляется взаимодействие с сервисом путем вызова необходимых операций (пример представлен на рис. 5.1). Процедура создания прокси-класса зависит от того, на базе каких технологий строится клиентское приложение. Если выбраны язык программирования C# и платформа .NET, построение клиента производится за счет вызова служебного программного средства svcutil.exe, распространяемого в составе платформы .NET, либо за счет создания ссылки на сервис в среде Microsoft VisualStudio.

```
public class FlowDriver
{
    static void Main(string [] args)
    {
        // создание клиента сервиса
        FlowSystemService service = new FlowSystemService();
```

```
        // заполние запроса на выполнение задания
        JobRequest jobRequest = new JobRequest
            {
                Script = File.ReadAllText("somescript.txt"),
                ScriptDataContext = "semplinput = 891-289-1244-9001-023"
            };
        // выполние запроса на выполнение задания
        Guid flowId = service.PushJob(jobRequest);
    }
}
```

Рисунок 5.1 – Пример обращения к сервису FlowSystemService

6. ВХОДНЫЕ ДАННЫЕ

Входными данными для КИ со стороны клиента является объект класса JobRequest, в котором содержится описание запроса пользователя на исполнение WF. Он включает в себя:

- текст скрипта на языке EasyFlow в строковом формате;
- контекст данных для запуска WF, содержащий ссылки в хранилище данных на данные, используемые в WF;
- настройки исполнения WF.

7. ВЫХОДНЫЕ ДАННЫЕ

Выходными данными для КИ WF является объект класса JobDescription, описывающий результат выполнения композитного приложения с подробной информацией о выходных данных каждого блока WF. Он включает в себя:

- информация о состоянии WF (завершен успешно / частично / с ошибкой и пр.);
- идентификаторы файлов в хранилище данных, полученных в результате работы WF;
- подробную информацию, описывающую историю запусков отдельных шагов.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

БД	База данных
WF	Workflow (поток заданий)
КИ	Компонент интерпретации WF
МИТП	Многопрофильная инструментально-технологическая платформа
LRWF	Long Running WF, workflow длительного исполнения

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

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