**Сущности на грани миров**

**Version 1.0**

**Status: preliminary**

**Michael Groozman**

**Nov 08, 2013**

**Revision History:**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Comments** |
| Nov/08/13 | M.Groozman | Document created |

**Сущности на грани миров**

В VWML имеется возможность использования методов java (в принципе методов любого объектно-ориентированного языка).

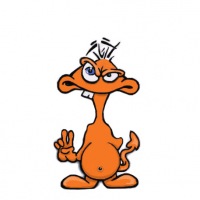
Отличительной особенностью встраивания конструкций java является возможность не особо задумываться над тем, что используемые в коде VWML конструкции являются методами объектов java. Т.е. код пишется полностью согласно синтаксису и семантики VWML. Информацию о том, что в этом коде является методами java объектов интерпретатор “черпает в настройках проекта”.

Мы можем представлять себе в самых общих чертах ситуацию так, как это изображено на рисунке 1.

Virtual World



****

fringe Bob

fringe

Robby

Java

Рис. 1.

Т.е. у нас имеются сущности, живущие на грани миров. Это fringe-сущности (контексты), каждая такая сущность и осуществляет связь Virtual World с миром Java.

Эти сущности удобно мыслить, как SapienZ, которые и “помогают интерпретатору разобраться”, где в тексте VWML идёт речь о методах Java и воспользоваться этими методами.

fringe-сущности декларируются в секции beyond настроек проекта.

Приведём пример этой секции.

beyond {

fringe Robby ias (

intmath ias < java package >

realmath ias < java package >

) ;

fringe Bob ias (

logic ias < java package >

) ;

}

Здесь Robby, Bob – сущности на грани миров.

Каждая из этих сущностей является контекстом в которых представлены объекты, которыми можно будет воспользоваться в VWML коде.

Точнее говоря, как в контексте Robby, так и в контексте Bob’a представлены псевдонимные сущности, которые интерпретируются как объекты Java.

В контексте Robby это:

intmath – математика целых чисел;

realmath – математика действительных чисел.

В контексте Bobа:

logic – объект, методы которого – логические операции.

Таким образом, Robby у нас математик, а Bob – логик.

Замечание:

Каждый fringe-контекст (fringe-сущность) состоит из последовательности интерпретационных выражений вида:

< псевдоним java объекта > ias < java package >

При этом вложение контекстов запрещено.

Переходим теперь к тому, как используются методы объектов java в VWML коде.

Мы предполагаем, что секция beyond, такова как было представлено выше.

Рассмотрим код простого проекта.

Example1 ias (

x ias 0 ;

y ias 1 ;

s ias nil ;

inc ias intmath.increment ;

Sum ias intmath.summation ;

action1 ias ( x

( x ↓ inc ↓ ) Do

) ↑ ;

action2 ias ( s

( ( x ↓ y ↓ ) Sum ↓ ) Do

) ↑ ;

LifeTerm = ( action1 ↓ exe

action2 ↓ exe

)

) / \* Example1 \*/

Рассмотрим интерпретационное выражение.

inc ias intmath.increment

псевдонимная воспринимается

сущность интерпретатором

метода как метод

increment java объекта

Таким образом в VWML коде inc ↓ используем как метод increment.

Итак, с точки зрения VWML inc ↓ - просто операция.

Рассмотрим теперь интерпретационное выражение.

action1 ias ( x

( x ↓ inc ↓ ) Do

) ↑

Здесь у нас встретилась новая операция Do языка VWML.

Операндом для неё является составная сущность

( x ↓ inc ↓ )

Интерпретация этой сущности даёт

( 0 intmath.increment )

Операция Do, по своему определению, заключается в применении метода intmath.increment к фактическому параметру 0. Получим в результате 1.

Интерпретация

action1 ↓ exe

модифицирует одно из интерпретационных выражений. Получим:

x ias 1 ;

Рассмотрим теперь интерпретационное выражение

Sum ias intmath.summation

как и ранее здесь Sum – псевдоним метода intmath.summation языка java.

Этот метод “умеет” складывать целые числа.

Sum ↓ с точки зрения VWML унарная операция, применяемая к составной сущности. Первая компонента которой – первое слагаемое, а вторая – второе слагаемое.

Таким образом,

( ( x ↓ y ↓ ) Sum ↓ ) Do

интерпретируется как применение операции

Sum ↓ к ( x ↓ y ↓ )

Перед интерпретацией терма:

action2 ↓ exe

мы имеем

x ias 1 ;

y ias 1 ;

Результат выполнения Do даст нам 2.

Результат интерпретации LifeTerma даст следующее “описание мира”.

x ias 1 ;

y ias 1 ;

s ias 2 .

Опишем теперь в общем виде операцию Do.

Do – унарная операция.

Её операндом является составной двухкомпонентный терм. Т.е. формат операции Do таков:

( < paramterm > < entity > ↓ ) Do

Здесь:

< paramterm > - неформально представляет собой список параметров

метода некоторого java-объекта.

Возможны 3 случая:

1). Параметры отсутствуют, тогда < paramterm > интерпретируется

как составная сущность без компонент - ( ) .

2). Имеется единственный параметр, тогда < paramterm >

интерпретируется как простая сущность или как однокомпонентная составная сущность.

3). Случай более одного параметра, тогда < paramterm >

интерпретируется как n-компонентная составная сущность.

< entity > - представляет собой псевдонимную сущность,

интерпретируемую как метод некоторого java-объекта.

Выполнение операции Do означает применение метода < entity > ↓ к аргументам, получаемым при интерпретации компонент списка < paramterm >.

Как обычно, результат операции Do вталкивается в стек.

Рассмотрим ещё один пример.

Секция beyond имеет вид:

beyond {

fringe Communicator ias (

Communicate ias < java package >

) ;

fringe RobbyMath ias (

intmath ias < java package >

) ;

}

Сам проект имеет вид:

Example2 ias (

helloMessage ias ( helloFriend

please find 1 plus 1 ) ;

x ias nil ;

ResultMessage ias ( Result is x ↓ ) ;

in ias Communicate.input ;

out ias Communicate.output ;

Sum ias intmath.GetSum ;

LifeTerm = (

( helloMessage ↓ out ↓ ) Do

( x

( ( 1 1 ) Sum ↓ ) Do ) ↑

( ResultMessage ↓ out ↓ ) Do

)

) /\* Example2 \*/

Предположим, что человек захотел, чтобы увеличили на 1 введённое им число. Тогда мы бы использовали:

1). Оператор Do для ввода

( x ( ( ) in ↓ ) Do ) ↑ ;

2). Ввели бы в рассмотрение псевдоним inc:

inc ias math.increment ;

и оператор Do:

( x ( x inc ↓ ) Do ) ↑ .