Дрождин В.В., Шалаев А.А. Язык разметки программ в самоорганизующейся информационной системе. // Проблемы информатики в образовании, управлении, экономике и технике: Сб. статей XIV Междунар. научно-техн. конф. – Пенза: ПДЗ, 2014. – С. 96-104.

УДК 004.436.4

ЯЗЫК РАЗМЕТКИ ПРОГРАММ В САМООРГАНИЗУЮЩЕЙСЯ ИНФОРМАЦИОННОЙ СИСТЕМЕ

В.В. Дрождин, А.А. Шалаев

PROGRAM MARKUP LANGUAGE IN A SELF-ORGANIZING INFORMATION SYSTEM

V.V. Drozhdin, A.A. Shalaev

Аннотация. В рамках нотации языка YAML предложен язык разметки программных модулей, позволяющий выделять функционально законченные блоки и связывать их между собой по иерархическому принципу. Это позволяет в случае необходимости находить неэффективные и некорректно работающие блоки (подзадачи и методы) и заменять их на логически эквивалентные, но имеющие лучшие характеристики по быстродействию и объемам ресурсов.

Ключевые слова: язык разметки, самоорганизующаяся информационная система, задача, спецификация задачи, метод решения.

Abstract. This paper proposes a YAML-based markup language of application modules. The language both defines functionally complete blocks and links them together in a hierarchical manner. If necessary, this allows to find inefficient and incorrectly running blocks (subtasks and methods) and to replace them with their logical equivalents that have better characteristics in terms of performance and resource usage.

Keywords: markup language, self-organizing information system, task, task specification, method of solution.

Самоорганизация информационной системы вызывает необходимость модификации программного кода в процессе исполнения программы. Программный код задает решаемые задачи, последовательность решения подзадач, обеспечивающих решение задачи, и методы, используемые для решения каждой подзадачи. Поэтому спецификация задач (подзадач) и методов их решения обеспечивает возможности изменения программного кода и существенно влияет на эффективность функционирования системы.

Задача – это точная спецификация решения некоторой проблемы, включающая требования (цель), условия (известное) и искомое (неизвестное).

Спецификация задачи (подзадачи):

$$z = <id_z, S_u, S_к, c, \mu>,$$

где id_z – идентификатор (имя) задачи, выделяющий ее среди множества всех других задач;

$S_u = \{s_u\}$ – область определения задачи (набор входных параметров, их содержание и допустимые значения, а также допустимые состояния входных данных);

$S_к = \{s_к\}$ – область значений задачи (набор выходных параметров, их содержание и допустимые значения, а также допустимые состояния выходных данных);

c – ограничения на задачу (ограничения на области определения и значений задачи, ограничения на быстродействие и объемы ресурсов, а также соответствие q_zмежду исходными и результирующими состояниями в форме отношения $q_z = \{\langle s_u, s_к\rangle\}$ или функции q_z: $s_u \rightarrow s_к$);

μ – метод решения задачи.

Метод решения задачи – это способ преобразования исходных состояний данных (известное) в выходные состояния данных (неизвестное) по определенному правилу (алгоритму), т.е. метод задает описание последовательности действий, определяющей процесс решения некоторой абстрактной задачи.

Спецификация метода решения задачи:

$$\mu : s'_u \xrightarrow{f} s'_к,$$

где $S'_u = \{s'_u\}$ – область определения метода (набор входных параметров, их содержание и допустимые значения, а также допустимые состояния входных данных);

$S'_к = \{s'_к\}$ – область значений метода (набор выходных параметров, их содержание и допустимые значения, а также допустимые состояния выходных данных);

f – правило (алгоритм) преобразования s'_u в $s'_к$.

При разработке языка описания задач и возможных методов их решения целесообразно учитывать следующие принципы:

- контекстную независимость конструкций языка;
- структурированность размечаемой программы;
- невидимость конструкций языка при трансляции программного кода;
- представление конструкций языка в виде тэгов;
- открытость языка.

Таким образом, конструкции языка разметки должны эффективно обрабатываться, а свойство открытости позволить системе расширять существующие конструкции новыми компонентами и включать новые конструкции с соответствующей семантикой.

Описание задачи задается с помощью тегов task, subtask, declaration, input, output, constraints, method и implementation. Тег task (subtask) инициирует описание задачи (подзадачи). Тег declaration содержит идентификатор и семантическое описание решаемой задачи (подзадачи), а также идентификатор класса в классификаторе задач, к которому относится решаемая задача. Теги input и output определяют допустимые комбинации входных и выходных данных. Тег constraints задает ограничения на процесс решения задачи и правила соответствия входных комбинаций данных выходным. Тег method указывает, что задача является простой и задает метод ее решения. Если в теге method не указан идентификатор метода, то в качестве метода, используемого для решения данной задачи, может использоваться любой метод решения данной задачи, указанный в классификаторе задач, или метод решения более общей задачи. Тег implementation указывает, что задача является сложной и описывает процесс ее решения в виде последовательности подзадач.

Описание метода решения абстрактной задачи задается с помощью тегов method, input, output, constraints и source. Тег method инициирует описание метода. Тег declaration содержит идентификатор и семантическое описание метода, а также идентификатор класса в классификаторе задач, к которому относится абстрактная задача, решаемая данным методом. Теги input и output определяют допустимые комбинации входных и выходных данных. Тег constraints задает ограничения на процесс решения абстрактной задачи и правила соответствия входных комбинаций данных выходным. Тег source содержит программный код, реализующий метод.

При конструировании процесса решения некоторой задачи осуществляется взаимосвязь описания данной задачи с подзадачами и методами по данным и управлению.

Связь по данным производится путем указания эквивалентности между входами задачи с соответствующими входами подзадач и выходов подзадач с соответствующими входами следующих подзадач или выходами задачи, а также входов и выходов подзадач с соответствующими входами подзадач более низкого уровня или реализующими их методами.

Связь по управлению в процессе решения задачи осуществляется путем последовательного размещения описаний подзадач или использованием управляющих операторов: условного, выбора или циклов.

В качестве примера приведем два варианта программных модулей, осуществляющих решение квадратного уравнения $y = ax^2 + bx + c$.

Для этого считаем, что в репозитории методов СИС имеются описания двух методов.

1. Эффективный метод решения задачи для случая $y = ax^2 + 2kx + c$.

```
method: {id: 1002, name: Решение квадратного уравнения – четный коэффициент b, class: 10}
input:
  – {param: pa, name: Коэффициент 1, component: basic, type: double}
  – {param: pb, name: Коэффициент 2, component: basic, type: double}
  – {param: pc, name: Коэффициент 3, component: basic, type: double}
inputPredicate:
  – pb % 2 == 0
output:
  – {param: r1, name: Корень 1, component: basic, type: double}
  – {param: r2, name: Корень 2, component: basic, type: double}
outputPredicate:
  – Null
constraints:
  characteristics: {performance: t = 0.001 s, memory: 1 kB, external: Null}
  tests:
    – input(3,-12,-135), output(9,-5)
    – input(1,-3,-4), output(-1,4)
source:
  var k = pb / 2;
  var discriminant = k * k – pa * pc;
  if (discriminant < 0)
    Console.WriteLine("No Real Solution!");
  else if (discriminant > 0)
  {
```

```
      var sqrt = System.Math.Sqrt(discriminant);
      r1 = (-k + sqrt) / pa;
      r2 = (-k – sqrt) / pa;
      Console.WriteLine("Two Real Solutions: x1 = {0:f3} || x2 = {1:f3}", r1, r2);
   }
   else
   {
      r1 = r2 = -k / pa;
      Console.WriteLine("One Real Solution: {0:f3}", r1);
   }
#method END 1002
```

2. Метод решения задачи для общего случая.

```
method: {id: 1001, name: Решение квадратного уравнения – общий случай, class: 10}
input:
   – {param: pa, name: Коэффициент 1, component: basic, type: double}
   – {param: pb, name: Коэффициент 2, component: basic, type: double}
   – {param: pc, name: Коэффициент 3, component: basic, type: double}
inputPredicate:
   – Null
output:
   – {param: r1, name: Корень 1, component: basic, type: double}
   – {param: r2, name: Корень 2, component: basic, type: double}
outputPredicate:
   – Null
constraints:
   characteristics: {performance: t = 0.002 s, memory: 1.5 kB, external: Null}
   tests:
      – input(3,-12,-135), output(9,-5)
      – input(1,-3,-4), output(-1,4)
source: |
   var discriminant = pb * pb – 4 * pa * pc;
   if (discriminant < 0)
      Console.WriteLine("No Real Solution!");
   else if (discriminant > 0)
   {
      var sqrt = System.Math.Sqrt(discriminant);
      r1 = (-pb + sqrt) / (2 * pa);
      r2 = (-pb – sqrt) / (2 * pa);
      Console.WriteLine("Two Real Solutions: x1 = {0:f3} || x2 = {1:f3}", r1, r2);
   }
   else
   {
      r1 = r2 = (-pb + System.Math.Sqrt(discriminant)) / (2 * pa);
      Console.WriteLine("One Real Solution: {0:f3}", r1);
   }
#method END 1001
```

Вариант 1 – решение квадратного уравнения – простая задача.

```
task:
declaration: {id: 1, name: Решение квадратного уравнения – простая задача, class: 10}
input:
   – {param: a, name: Коэффициент при x^2, component: basic, type: double}
   – {param: b, name: Коэффициент при x, component: basic, type: double}
   – {param: c, name: Коэффициент при x^0, component: basic, type: double}
```

```
inputPredicate:
  – Null
output:
  – {param: x1, name: Корень x1, component: basic, type: double}
  – {param: x2, name: Корень x2, component: basic, type: double}
outputPredicate:
  – Null
constraints:
  requirements: {performance: t = 0.001 s, memory: < 2 kB, external: Null}
  tests:
    – input(3,-12,-135), output(9,-5)
    – input(1,-3,-4), output(-1,4)
method: Null
#taskEND 1
```

Так как в теге method не указан идентификатор конкретного метода, то при построении программного модуля может использоваться эффективный метод или метод для общего случая, в зависимости от того, какие данные ожидаются на входе решаемой задачи.

Вариант 2 – решение квадратного уравнения – сложная задача.

```
task:
declaration: {id: 2, name: Решение квадратного уравнения – сложная задача, class: 10}
input:
  – {param: a, name: Коэффициент при x^2, component: basic, type: double}
  – {param: b, name: Коэффициент при x, component: basic, type: double}
  – {param: c, name: Коэффициент при x^0, component: basic, type: double}
inputPredicate:
  – Null
output:
  – {param: x1, name: Корень x1, component: basic, type: double}
  – {param: x2, name: Корень x2, component: basic, type: double}
outputPredicate:
  – Null
constraints:
  requirements: {performance: t = 0.001 s, memory: < 2 kB, external: Null}
  tests:
    – input(3,-12,-135), output(9,-5)
    – input(1,-3,-4), output(-1,4)
implementation:
  if (b % 2 == 0)
    {
     subtask:
     declaration: {id: 901, name: Решениеквадратногоуравнениясчетнымb}
     input:
       – {param:a, name:Коэффициент 1, component:basic, type:double, equalTo:101.a}
       – {param:b, name:Коэффициент 2, component:basic, type:double, equalTo:101.b}
       – {param:c, name:Коэффициент 3, component:basic, type:double, equalTo:101.c}
     output:
       – {param: r1, name: Корень 1, component: basic, type: double, equalTo: 101.x1}
       – {param: r2, name: Корень 2, component: basic, type: double, equalTo: 101.x2}
     method: 1002
     #subtaskEND 901
     }
     else
```

```
{
subtask:
declaration: {id: 902, name: Решениеквадратногоуравнения}
input:
  – {param:a, name:Коэффициент 1, component:basic, type:double, equalTo:101.a}
  – {param:b, name:Коэффициент 2, component:basic, type:double, equalTo:101.b}
  – {param:c, name:Коэффициент 3, component:basic, type:double, equalTo:101.c}
output:
  – {param: r1, name: Корень 1, component: basic, type: double, equalTo: 101.x1}
  – {param: r2, name: Корень 2, component: basic, type: double, equalTo: 101.x2}
method: 1001
#subtaskEND 902
}
#taskEND 2
```

Использование для решения квадратного уравнения двух методов, представленных подзадачами 901 и 902, будет более эффективно вычислять корни уравнения, если входные данные достаточно часто содержать четный коэффициент b.

Таким образом, в рамках нотации языка YAML [1], поддерживаемых на платформе .NET библиотекой YamlDotNet [2], предложен язык разметки программных модулей, позволяющий выделять функционально законченные блоки и связывать их между собой по иерархическому принципу, а в случае необходимости некоторые блоки (подзадачи или методы) могут быть заменены на эквивалентные, но имеющие лучшие характеристики по быстродействию и объемам ресурсов.

Библиографический список

1. Язык разметки YAML. – URL: http://yaml.org/ (дата обращения: 05.10.2014).

2. Библиотека YamlDotNet для YAML на платформе .NET. – URL: https://github.com/aaubry/YamlDotNet (дата обращения: 05.10.2014).

Дрождин Владимир Викторович
Пензенский государственный университет, г. Пенза, Россия
E-mail: drozhdin@yandex.ru

Drozhdin Vladimir Viktorovich
Penza State University,
Penza, Russia

Шалаев Александр Александрович
Пензенский государственный университет, г. Пенза, Россия
E-mail: drozhdin@yandex.ru

Shalaev Aleksandr Aleksandrovich
Penza State University,
Penza, Russia