Введение в Cyberiada-GraphML

Версия: 1.0 (14 августа 2024 г.)

Редакторы:

- Михаил Чекан, tg: @chekoopa
- Алексей Федосеев, tg: @afedoseev

Аннотация

Данный документ содержит неформализованное описание (primer) Cyberiada-GraphML, формата для описания расширенных иерархических машин состояний, совместимых с НКФП.ПРИМС и рассчитанных для исполнения на той или иной целевой платформе управления. Задача документа – изложить основы формата и базовый набор расширений интеграции с программными средствами и применения в практических задачах.

Содержание

1 Введение

Формат Cyberiada-GraphML (сокращённо CGML) предназначен для представления в текстовом виде расширенных иерархических машин состояний (РИМС), совместимых со стандартом программирования РИМС[1] (версия 1.0) в рамках Национальной киберфизической платформы (далее НКФП.ПРИМС). В формат закладывается с одной стороны разделение на целевые платформы, под которой предполагается функционирование описанной РИМС, с другой – набор расширений, задействуемых в зависимости от выбранной целевой платформы.

Документ нацелен на разработчиков приложений с машинами состояний, желающих использовать формат для описания логики этих машин, а также разработчиков редакторов или иных приложений, визуализирующих или конвертирующих схемы в иной формат.

При всём намерении предоставить максимально доступное описание формата, предполагается, что читатель владеет хотя бы на базовом уровне форматом XML и имеет представление о пространствах имён в XML. Знание GraphML не требуется, но поможет быстрее понять описываемый формат. Представление о парадигме ПРИМС необходимо для полного понимания документа, но авторы стремились дать максимально доступное краткое описание связанных концепций.

Раздел 2 приводит базовые концепции, на которые опирается описываемый формат.
Раздел 3 и 4 содержит ядерные элементы формата, наследуемые из GraphML и формирующие граф машины состояний с описанием событий и действий.
В раздел 5 раскрывается текстовый формат, используемый в совокупности с языком целевой платформы для описания сигналов и действий в событиях и переходах.
Раздел 6 перечисляет расширения формата, относящиеся к стандартным, т.е. предлагаемые «из коробки».

[1] https://docs.google.com/document/d/14ptmhxw8529gtEFLlNu3IpZ3DhIo0DImQ5RjdHzxm8A

2 Базовые концепции

Формат описывает **расширенную иерархическую машину состояний** (РИМС). Ёмкий обзор РИМС приводится в разделе 2 стандарта НКФП.ПРИМС, здесь же далее приводится краткая сводка иными словами.

Машина состояний (конечный автомат) – модель дискретного устройства, имеющего вход (для входных сигналов или событий), выход (для выходных воздействий или поведения) и конечное число состояний, в одном из которых устройства пребывает в каждый момент времени. Состояния и переходы между ними образуют ориентированный граф.

Иерархичность означает, что состояния могут быть вложенными друг в друга, и переход во вложенное состояние также активирует родительское состояние, задействуя связанные с ним реакции на события. В зависимости от конфигурации устройства определяются детали, например, активируется ли какое-либо дочернее состояние при активации родительского, задействуется ли что-то кроме переходов из родительского состояния и т.д.

Расширенность означает, что модель РИМС сочетает в себе автомат Мили, где выходные воздействия формируются при переходах, и автомат Мура, где выходные воздействия привязаны к состояниям. Так, для РИМС характерны переходы по событиям как в другие состояния, так и в само себя, при этом каждый из этих переходов может сопровождаться выполнением поведения (набор выходных воздействий). Кроме этого, в РИМС предусмотрены события entry и exit, соответственно срабатывающие при переходе в состояние (имитируя логику Мура) и при выходе из состояния.

Пример РИМС, взятый из игры «Академия» (стандартное поведение робота-пылесоса) платформы «Берлога» приведён на рисунке 1.

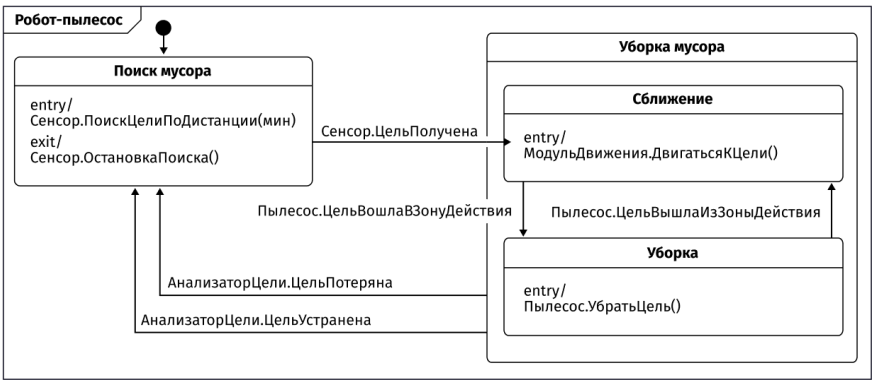


Рисунок 1 – Пример диаграммы состояний из игры «Академия»

Таким образом, машина имеет конечный набор **состояний**. Состояния могут вкладываться друг в друга, образуя древовидную иерархию для выражения подпроцессов. Состояние, в которое вложено одно или более состояний, называется

надсостоянием или родительским состоянием. Состояние, вложенное в другое, называется подсостоянием или дочерним состоянием.

Переход между состояниями осуществляется по возникновении **событий**, на которые также можно накладывать **ограждающие условия** в виде предикатов и сравнений. Переход в определённое состояние активирует все родительские ему состояния, а переходы родительского состояния действительны для всех активных состояний, вложенных в него. При этом кроме переключения состояния предусмотрено выполнение **поведения** (набора **действий**). Переход может не влечь смену состояния, тогда считается, что действие выполняется в рамках текущего состояния.

Для всех состояний предусмотрены «общие» события – **entry** (действия при активации состояния) и **exit** (действия при выходе из состояния). Строго говоря, это не отдельные события, а общие наборы действий, подставляемые в переходы между состояниями и предусмотренные для устранения избыточности.

Для формирования сложных переходов и обозначения начала выполнения состояний применяются **псевдосостояния**, отображаемые на диаграмме в виде точек. **Начальное псевдосостояние** указывает на **начальное состояние**, с которого начинается выполнение программы. В зависимости от набора расширений входное состояние будет единственным, или таких будет несколько (по одному на каждый набор вложенных состояний). Дополнительно РИМС может содержать **конечное состояние**.

Документ – совокупность машин состояний, объединённых по смыслу или назначению в один файл. Соответственно, Cyberiada-GraphML описывает формат документа, редактируемого, визуализируемого, интерпретируемого (в частности для описания поведения объектов внутри программы), а также транслируемого в другие языки программирования для использования в прикладных сценариях.

Представление и семантика РИМС подробно описаны в НКФП.ПРИМС, и в своей основе соответствуют машинам состояний в UML, более подробно описанным в *Samek M. Practical UML statecharts in C/C++: event-driven programming for embedded systems. – CRC Press, 2008* (далее – PSiCC2).

3 Базовый формат

Как следует из названия, в основе Cyberiada-GraphML лежит формат представления графов GraphML, и за вычетом расширений схема представляет корректный файл данного формата. Это позволяет применять для визуализации и обработки (второе частично) уже существующие распространённые программные решения с поддержкой этого формата.

Базовый формат GraphML используется для определения структуры элементов и связей. За это отвечают теги *graph*, *node* и *edge*. Далее описывается базовый синтаксис этих тегов, используемый в GraphML и применяемый в дальнейшем для описания элементов CGML.

3.1 Документ

В основе GraphML (и соответственно CGML) находится формат XML с простым формальным синтаксисом, удобным для создания и обработки документов как программами, так и человеком. Документ в таком формате является текстовым файлом в кодировке UTF-8. Использование других кодировок допустимо в зависимости от обстоятельств применения и особенностей целевой платформы, но не рекомендуется в общем случае.

В основе документа находится корневой элемент GraphML. В минимальном виде он выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
…
</graphml>
```

Это можно сравнить с корневым тегом *html* и предшествующим ему *DOCTYPE*, объявляющим версию языка. Атрибуты тега *graphml* содержат объявления пространств имён, используемых в схеме. Конкретно атрибут *xmlns* объявляет пространство тегов GraphML, необходимых для формата. В зависимости от используемых расширений тег может дополняться иными атрибутами.

3.2 Низкоуровневые комментарии

Родительский формат XML предусматривает низкоуровневые комментарии в тексте документа. Соответственно, CGML также допускает их использование при работе напрямую с текстовым представлением документа (например, чтобы оставлять замечания или временное удалять сегменты), а также для машиночитаемых пометок. Синтаксис комментария идентичен используемому в XML и HTML, блок с маркером начала <!-- и маркером конца -->:

```
<!-- Это комментарий в Cyberiada-GraphML-документе,
и он может быть многострочным, потому что как бы мы иначе
прятали блоки текста -->
```

Не следует путать низкоуровневые комментарии (в рамках XML-разметки) и комментарии диаграмм машин состояний, описанные в стандарте ПРИМС и далее в этом документе (раздел 4.9).

3.3 Метка формата

Для идентификации файла как документа в формате CGML сразу после открывающего тега *graphml* необходимо разместить **метку формата** в виде элемента *data* с атрибутом *id="gFormat"* и фиксированным значением *Cyberiada-GraphML-1.0*:

```
<data key="gFormat">Cyberiada-GraphML-1.0</data>
```

3.4 Ключи для полей данных

Одной из задач формата CGML ставится совместимость с GraphML, и это является причиной минимизации собственных тегов в формате. Основа формата GraphML описывает структуру элементов и связей, для описания же их полезной нагрузки предусмотрен тег *data*, сопровождаемый атрибутом *key* (далее – ключ или типирующий ключ), который идентифицирует тип содержимого тега. Формат предписывает объявлять используемые ключи, привязывая их к тегам, к которым они прикладываются, и тип содержимого. Это подразумевает наличие **объявлений**. Они указываются указываются между меткой формата и остальным содержимым документа, и имеют следующий вид:

```
<key id="dName" for="node" attr.name="name" attr.type="string"/>
```

Обработка содержимого РИМС в формате производится исключительно по ключу. Наличие ключа в теге *data* обязательно, и именно ключ определяет тип содержимого для обработки. Описанные в данном документе ключи являются основой формата, их переопределение недопустимо. Но при этом допускается приложениям и редакторам вводить свои ключи при необходимости (например, в рамках расширений).

Полный перечень объявлений доступен в приложении Б. Объявления несут преимущественно справочную цель, их содержимое несёт рекомендательный характер и направлено на поддержку совместимости с GraphML-приложениями, использующими атрибуты (а здесь определяются их имя и тип). Следовательно, их наличие в документе рекомендуется, но при крайней необходимости (например, сокращение размера документа) блок объявлений можно опускать. Изменение имени и типа атрибутов для представленных в данном документе объявлений недопустимо.

Далее по тексту будет использоваться конструкции «*X с ключом Y*». Если тег X – не тег *data*, то подразумевается тег, соответствующий элементу X, в который вложен тег data

с атрибутом key, равным Y, если не указано иначе. Аналогично, «*значение ключа Y*» – это содержимое тега data с атрибутом *key*, равным Y.

3.5 Подграф

Тег graph описывает **граф или подграф** и может содержать узлы, рёбра и другие подграфы (через узел).

```
<graph id="gMain" edgedefault="directed">
…
</graph>
```

К тегу *graph* применяется атрибут *id*, обозначающий **уникальный идентификатор** (см. далее) элемента, например, *G*. Рекомендуется для идентификаторов подграфов использовать префикс *g*, после которого пишется содержательное название-идентификатор, либо ставится порядковый номер.

Атрибут *edgedefault* определяет ориентацию графа в области. РИМС предполагает ориентированный граф состояний, поэтому значение атрибута должно равняться *directed*. Но вне зависимости от значения атрибута, формат GraphML требует, чтобы все связи были направленными (см. далее), и это требование наследуется CGML.

3.6 Элемент-узел

Элементы машины состояний, являющиеся **узлами** (состояния, псевдосостояния, комментарии и т.д.) описываются тегом *node*, вложенным в *graph*. К тегу применяется обязательный атрибут id, содержащий **уникальный идентификатор** элемента.

```
<node id="n0">
…
</node>
```

Внутри тега размещаются данные, связанные с элементом, и определяющие его полезную нагрузку.

Рекомендуется для идентификаторов узлов использовать префикс *n*, после которого пишется содержательное название-идентификатор, либо ставится порядковый номер.

3.7 Элемент-ребро

Элементы машины состояний, являющиеся **ребрами** (переходы и связи), описываются тегом *edge*, вложенным в *graph*. К тегу также применяется обязательный атрибут id, содержащий **уникальный идентификатор** элемента. Кроме этого, к тегу применяются

обязательные атрибуты *source* и *target*, отражающие соответственно **исходящий** и **входящий** узлы связи, выраженные их уникальными идентификаторами.

```
<edge id="n0-n3#0" source="n0" target="n3">
  ...
</edge>
```

Внутри тега размещаются данные, связанные с ребром, и определяющие его полезную нагрузку.

Между двумя узлами может быть любое число рёбер, в том числе переходов. Также возможно создание ребра из узла в само себя.

Рекомендуется составлять идентификаторы рёбер из идентификаторов исходящего и входящего узлов, добавляя к ним порядковый перехода во избежание коллизий: *source-target#0*.

Размещать теги рёбер следует **единым блоком** в конце узла машины состояний (см. 4.1), к которой рёбра принадлежат.

3.8 Идентификаторы

Все структурные элементы документа обязаны иметь атрибут *id* (далее – идентификатор), содержащий **идентификатор** элемента. Идентификатор может содержать последовательность символов произвольной длины, в том числе подчёркивания и двоеточия.

Идентификатор элемента должен быть уникален относительно других элементов **документа** (не только в пределах МС). В этом отношении идентификатор можно считать квалифицированным именем в **глобальном** пространстве имён документа. В частности, идентификатор может быть задействован расширениями (например, при использовании параллельных или ортогональных МС).

4 Ядро описания РИМС

Данный раздел приводит содержательные надстройки над GraphML, определяющие формат CGML и предназначенные для описания полезной нагрузки РИМС. Здесь они приводятся в порядке от базовых концепций к частным, опираясь на перечисленные ранее.

4.1 Машина состояний

Тег *graph* на родительском уровне (вложенный в *graphml*) с ключом *dStateMachine* определяет машину состояний, отдельную совокупность элементов МС.

```
<graph id="G" edgedefault="directed">
  <data key="dStateMachine"/>
  <data key="dName"/>Посадка на Луну</data>
…
</graph>
```

Ключ *dStateMachine* **обязан** быть **первым** дочерним элементом тега *graph*. Значение ключа должно быть пустым, т.к. ключ является маркером типа узла.

Кроме ключа *dStateMachine*, тег *graph* МС содержит другие ключи, определяющие название и параметры МС (см. далее), а также непосредственно элементы, принадлежащие этой МС. Эти компоненты могут приводиться внутри тега в любом порядке, но наличие упорядоченности способствует более простой работе с файлом со стороны человека.

Допустимо приведение нескольких тегов на родительском уровне для описания связанных или параллельно функционирующих машин состояний, если это предусмотрено целевой платформой. Текущая версия стандарта не подразумевает наличие более одной области внутри машин состояний и составных состояний.

4.2 Простое состояние

Узел с ключом *dData* без иных типирующих ключей определяет состояние МС.
Тег *node* прописывается внутри машин состояний и составных состояний (в *graph).*

```
<node id="nScan">
  <data key="dData">entry/
Сенсор.ПоискЦелиПоДистанции(мин)
 <!-- … -->
exit/
Сенсор.ОстановкаПоиска
</data>
</graph>
```

Содержимое ключа *dData* определяет внутренние переходы данного состояния (см. далее).

4.3 Переход

Ребро с ключом *dData* без иных типирующих ключей определяет переход МС. Тег *edge* прописывается внутри машин состояний и составных состояний (в *graph*).

```
<edge id="edge0" source="n0" target="n3">
  <data key="dData">
Сенсор.ЦельПолучена [Счетчик.ТекущееЗначениеСчетчика &gt;= 2]/
  </data>
</edge>
```

Содержимое ключа *dData* определяет содержание перехода, а именно событие, ограждающее условие и поведение при переходе (см. далее).

4.4 Имя

Для более наглядного представления предусмотрено добавление человекочитаемого **имени** узла или машины состояний. Внутри узла или подграфа прописывается не более одного тега *data* с ключом *dName* следующего вида:

```
<node id="Scan">
  <data key="dName">Скан</data>
</node>
```

Отсутствие этого тега допустимо, в таком случае элемент имеет пустое имя или не обладает именем вовсе (но всё ещё определяется через уникальный идентификатор).

4.5 Геометрия

Ключ *dGeometry* используется для определения визуальных положения и размеров элементов при отображении и редактировании документа. Граф МС не обязан содержать геометрию (например, для случая машино-читаемых описаний МС). Ключ геометрии прописывается в узлах и рёбрах, и внутри прописываются теги *point* и *rect*, аналогичные используемым в SVG для обозначения соответственно точки и прямоугольника:

```
<data key="dGeometry">
  <point x="640" y="250"/>
</data>
<data key="dGeometry">
  <rect x="-630.2711" y="206.705933" width="468" height="170"/>
```

```
</data>
```

Координаты заданы на графической плоскости (аналогично SVG), ось Y направлена вниз. Параметры *x* и *y* задают координату, для *rect* – это координата левого верхнего угла элемента, для *point* – координата центра. При их отсутствии *x* или *y* соответствующее значение считается нулевым. Параметры *width* и *height* используются для определения размеров состояний и комментариев, они являются обязательными.

Все величины указываются в вещественных числах и считаются **относительно** левого верхнего угла элемента, родительского для того, где прописана геометрия. Значение этих величин может ограничиваться аппаратными характеристикам платформы редактора. Но на целевую платформу эти величины не влияют, т.к. не относятся к управляющей части РИМС.

У машин состояний, состояний и комментариев геометрия задаётся одиночным прямоугольником, а у псевдосостояний и конечного состояния – одиночной точкой.

Геометрия также применима для определения линии перехода, а именно задаёт промежуточные точки составной линии (относительно верхнего левого угла родительского состояния или машины состояний):

```
<data key="dGeometry">
  <point x="640" y="250"/>
  <point x="640" y="300"/>
  <point x="690" y="300"/>
</data>
```

В частном случае составная линия может содержать одну точку.

4.6 События и поведение

Главный элемент как состояния, так и перехода – события и поведение, отражающие функциональность машины состояний. Содержание зависит от целевой платформы и по сути представляет собой код на языке этой платформы, разбитый на переходы. Более подробно содержание этого элемента рассматривается в *разделе 5*.

Соответственно, в узлах и рёбрах прописываются ключи *dData* с описанием событий и поведения. Обратите внимание, что при описании поведения в состояниях и событиях используются пустые строки (обозначены в тексте как ↳). Они являются частью синтаксиса (см. раздел 5.2). В остальных случаях пустые строки добавлены для читаемости.

```
<node id="Scan">
  <data key="dData">entry/
Сенсор.ПоискЦелиПоДистанции(мин)
```

```
Сенсор.ЦельПолучена/
МодульДвижения.ЗадатьКоординатуВверх(34)
↳
exit/
Сенсор.ОстановкаПоиска
  </data>
</node>

<edge id="edge0" source="n0" target="n3">
  <data key="dData">
Сенсор.ЦельПолучена [Счетчик.ТекущееЗначениеСчетчика &gt;= 2]/
  </data>
</edge>
```

При отсутствии ключа *dData* в случаях, где он используется, его значение считается пустым. Так, пустое ребро будет рассматриваться как связь в зависимости от контекста (релевантно для расширений, но в текущей версии такое ребро не несёт полезной нагрузки), а пустой узел – простое состояние без названия и внутренних событий.

4.7 Составное состояние

Если состояние предусматривает вложенные состояния и переходы, в узел вкладывается подграф, внутрь которого складываются другие узлы и переходы.

```
<node id="nClean">
  <data key="dData"></data>

  <graph id="nClean:">
    <node id="nClean:nApproach">
      <data key="dData"></data>
    </node>
    <node id="nClean:nClean">
      <data key="dData"></data>
    </node>
    <edge id="nClean:nClean-nApproach"
          source="nClean:nApproach" target="nClean:nClean">
      <data key="dData"></data>
    </edge>
  </graph>

</node>
```

В качестве идентификатора подграфа и префикса идентификаторов вложенных состояний рекомендуется использовать идентификатор родительского состояния с

разделителем в виде двоеточия, после которого пишется собственный идентификатор состояния.

4.8 Вершины (псевдосостояния и конечное состояние)

Псевдосостояния – переходные вершины, используемые для создания составных переходов, а также для обозначения потока выполнения машины состояний (например, начальное псевдосостояние). Функционально близким к псевдосостояниям является конечное состояние, обозначающее завершение выполнения охватывающей его области. Обусловимся называть их **вершинами**.

Для обозначения вершин используется узел с ключом *dVertex*. Значение ключа определяет тип вершины (см. таблицу ниже). Стоит заметить, если не указано иначе, в узле рядом с *dVertex* не используются ключи *dData*, при наличии их содержимое игнорируется. Использование *dName* допустимо и интерпретируется как имя вершины (отображаемое как подпись). Но в этом и иных случаях ключ *dVertex* обязан быть **первым** дочерним элементом узла.

Тег	**Тип вершины**	**Примечание**
`initial`	начальное псевдосостояние	
`final`	конечное состояние	
`choice`	псевдосостояние выбора	
`terminate`	завершающее псевдосостояние («исключение»)	
`shallowHistory`	псевдосостояние локальной истории	*extra*
`deepHistory`	псевдосостояние глубокой истории	*extra*
`entryPoint`	точка входа	*extra*
`exitPoint`	точка выхода	*extra*
`fork`	разветвление	*future*
`join`	объединение	*future*

Приписка *extra* означает, что эти типы вершин предусмотрены в стандартных расширениях формата (т.е. их наличие зависит от конкретного приложения и платформы). Приписка *future* означает, что указанные типы вершин существуют в стандарте, но зарезервированы для следующих версий формата.

4.8.1 Начальное псевдосостояние

Для обозначения начального псевдосостояния используется узел, с ключом *dVertex*, равным *initial*.

```
<node id="init">
  <data key="dVertex">initial</data>
</node>
```

По идентификатору начальное псевдосостояние соединяется с состоянием, являющимся начальным в данном графе или подграфе. В ключе *dData* перехода можно указать поведение, производимое целевой платформой в начале выполнения программы. Действия указываются как есть, без события и ограждающих условий.

```
<edge id="init-Scan" source="init" target="Scan">
  <data key="dData">Сенсор.ПоискЦелиПоДистанции(мин)
Таймер.ТаймерЗапуск(10)
  </data>
</edge>
```

В подграфе на родительском уровне должно присутствовать не более **одного** начального псевдосостояния, из которого должен идти ровно **один** переход в узел в пределах этого подграфа (другими словами, ссылка на узел вне подграфа некорректна).

В частном случае начальное псевдосостояние может существовать в единственном экземпляре и размещаться в родительском графе, тогда оно может ссылаться на узлы дочерних подграфов. Отсутствие псевдосостояния указывает на отсутствие входной точки автомата, что некорректно для родительского графа, но может иметь место во вложенных. Поведение при отсутствии дочернего начального псевдосостояния определяется реализацией целевой платформы.

4.8.2 Конечное состояние

Для обозначения конечного состояния используется ключ *dVertex*, равный *final*.

```
<node id="end">
  <data key="dVertex">final</data>
</node>
```

Далее к этому узлу как целевому приводится неограниченное число ребер-переходов с событиями. Использование данного узла в качестве источника перехода некорректно.

4.8.3 Завершающее состояние

Для обозначения завершающего состояния (аварийного завершения МС) используется ключ *dVertex*, равный *terminate*.

```
<node id="crash">
  <data key="dVertex">terminate</data>
  <data key="dName">Нехватка памяти</data>
</node>
```

Далее к этому узлу как целевому приводится неограниченное число ребер-переходов с событиями. Использование данного узла в качестве источника перехода некорректно.

4.8.4 Псевдосостояние выбора

В стандарте ПРИМС существует псевдосостояние выбора, в которое входит более одного перехода с событием, и из которого выходят переходы с ограждающими условиям и поведением. Это позволяет строить составные переходы, выражающие ветвление в зависимости от условий.

```
<node id="nChoice">
  <data key="dVertex">choice</data>
</node>

<edge id="Init-nChoice" source="Init" target="nChoice">
  <data key="dData">ИнициализацияВыполнена [не ЕстьОшибки]/
  </data>
</edge>
<edge id="nChoice-nWarn" source="nChoice" target="nWarn">
  <data key="dData">[ЧислоПредупреждений &gt; 0]/Диод.Зажечь(желтый)
  </data>
</edge>
<edge id="nChoice-nIdle" source="nChoice" target="nIdle">
  <data key="dData">[else]/
  </data>
</edge>
```

Далее к этому узлу как целевому приводится неограниченное число ребер-переходов с событиями (допустимо использование ограждающих условий), из узла приводится неограниченное число рёбер к другим состояниям и вершинам (в том числе другим псевдосостояниям выбора). Переходы из псевдосостояний выбора **обязательно** содержат ограждающее условие (в том числе специальное условие *else*) и могут сопровождаться поведением (которое выполнится при исполнении составного перехода).

В этой версии формата условия прописываются исключительно в рёбрах. Стандарт НКФП.ПРИМС предусматривает упрощённое обозначение для бинарных выражений с общим левым операндом, но в формате CGML текущей версии данное обозначение не предполагается.

4.9 Комментарий

Для пометок и примечаний в схеме используются узлы с произвольными идентификаторами, ключом *dNote* и данными по ключу *dData*. Такие узлы считаются комментариями, и ключ dNote определяет тип комментария:

- *informal* – человекочитаемый (неформальный, *по умолчанию*)
- *formal* – машиночитаемый (формальный)

Отсутствие значения в ключе *dNote* соответствует человекочитаемому типу, но настоятельно рекомендуется указывать значение ключа явно.

В комментариях ключ *dNote* **обязан** быть **первым** дочерним элементом узла. Если комментарий имеет заголовок, ключ *dName* идёт непосредственно **следующим** за *dNote* элементом.

```
<node id="noteX">
  <data key="dNote">informal</data>
  <data key="dName">TODO</data>
  <data key="dGeometry">
    <rect x="0" y="50" width="200" height="100"/>
  </data>
  <data key="dData">Когда-нибудь мы это реализуем.
Но не сегодня.</data>
</node>

<node id="noteY">
  <data key="dNote">formal</data>
  <data key="dName">gcc-arguments</data>
  <data key="dGeometry">
    <rect x="250" y="100" width="300" height="100"/>
  </data>
  <data key="dData">-O3 -s -o my_stateful_program</data>
</node>
```

4.10 Предмет комментирования

Комментарий связывается с другими элементами РИМС через связи, ведущие к комментируемым узлам. Другими словами, ребро от комментария к любому другому элементу является связью комментария с этим элементом.

Для более подробного указания предмета комментирования применяется ключ *dPivot*, содержащий компонент целевого элемента, на содержимое которого ссылается комментарий: *dName* или *dData*. Пустое значение *dPivot* интерпретируется как указание на элемент целиком.

В связях между комментариев ключ *dPivot* **обязан** быть **первым** дочерним элементом ребра.

Для уточнения фрагмента текста используется ключ *dChunk*, содержащий комментируемый фрагмент текста.

```
<node id="noteX">
  <data key="dNote">informal</data>
  <data key="dData">Так надо!</data>
</node>

<edge id="edge0" source="n0" target="n3">
  <data key="dData">
Сенсор.ЦельНайдена [Счетчик.ТекущееЗначениеСчетчика &gt;= 2]/
  </data>
</edge>

<edge id="noteX-edge0" source="noteX" target="edge0">
  <data key="dPivot">dData</data>
  <data key="dChunk">ЦельНайдена</data>
</edge>
```

4.11 Метаданные

Формальный комментарий с заголовком CGML_META считается узлом, содержащим **метаданные** документа. Такое решение обусловлено совместимостью с существующими редакторами GraphML для более удобного доступа к настройкам документа.

Узел метаданных является единственным в документе, размещается на верхнем уровне первой машины состояний и не привязан ни к каким другим элементам, кроме другим комментариев.

Такой узел оформляется как узел с данными под ключами *dNote*, где указывается значение *formal*, и *dData*, в котором приводятся **метапараметры**, представляющие собой набор текстовых блоков вида: *название/ значение* (пробелы и табуляция после косой черты игнорируются). Значение может занимать несколько строк, но вне зависимости от их числа, параметры разделяются **пустой строкой**.

Состав метапараметров строго не регламентируется, в качестве стандартных предлагаются:

- *standardVersion* – версия стандарта ПРИМС (*обязательный параметр, на момент публикации равен* `1.0`)
- *platform* – идентификатор целевой платформы (отсутствие платформы означает, что документ является иллюстративным)
- *platformVersion* – версия целевой платформы
- *platformLanguage* – язык целевой платформы
- *target* – название целевой управляемой системы
- *name* – человекочитаемое название документа
- *author* – имя автора документа
- *contact* – контактные данные (почта, ссылка на соцсети)
- *description* – описание документа
- *version* – версия, ревизия документа
- *createdAt* – дата и время создания документа в ISO-формате (в UTC)
- *transitionOrder* – порядок выполнения действий
    - *actionFirst* – сначала поведение в переходе (по умолчанию)
    - *exitFirst* – сначала поведение при выходе из состояния
- *eventPropagation* – порядок обработки передаваемых событий
    - *block* – прерывать на первом же обработчике (по умолчанию)
    - *propagate* – передавать дальше
- *markupLanguage* – язык разметки неформальных комментариев по умолчанию (см. раздел 6.4)

В случае отсутствия метапараметра его значение считается пустым. Для вышеперечисленных необязательных метапараметров это допустимо.

Если в значении метапараметра используется символ /, его необходимо экранировать обратной косой чертой (т.е. \/).

Список метапараметров и их обязательность может менять в зависимости от целевой платформы и используемых расширений.

```
<node id="coreMeta">
  <data key="dNote">formal</data>
  <data key="dName">CGML_META</data>
  <data key="dData">platform/BearsTowerDefence
↳
standardVersion/ 1.0
↳
name/Важный документ
↳
author/Матросов В.М.
↳
contact/v.m.matrosov@mail.ru
↳
description/ Пример описания документа,
который может быть многострочным, потому что так удобнее
```

```
target/ Autoborder
  </data>
</node>
```

5 Ядро описания переходов

В описании переходов прописывается функциональное содержание машины состояний. По стандарту ПРИМС (которому соответствует данный формат) это содержание выражается совокупностью **переходов**, вызываемых **событиями**, на которые могут накладываться **условия**, и которые приводят к выполнению поведения. Всё это описывается в текстовом виде согласно принятой структуре, которая и приводится в данном разделе. Для удобства элементы описываются от частного к общему.

5.1 Событие

Событие – это сигнал, возникающий в целевой платформе при наступлении определённой ситуации и фиксируемый в схеме как событие (см. далее 5.4). Формат предполагает, что событие является именованным объектом, т.е. имеет вид обозначения константы, корректного для языка платформы. В общем виде это алфавитно-цифровая строка с подчёркиваниями, начинающаяся с буквенного символа. Также допустимо использование пробелов и точки для компонентно-ориентированных платформ.

Например:

```
Сенсор.ЦельНайдена
Timer1.timeout
TIMER_TICK
```

В соответствии с НКФП.ПРИМС, в качестве собственных событий целевой платформы нельзя использовать следующие ключевые слова: *entry*, *exit* и *do*. Эти слова используются для определения соответственно поведения входа, поведения выхода и поведения исполнения (последнее при наличии такой возможности в целевой платформе).

5.2 Поведение

Поведение согласно стандарту представляет собой участок кода на языке целевой платформы. Формат придерживается этого определения, и **блок поведения** может содержать любой текст, состоящий из последовательно идущих строк **без пропусков**. **Пустая строка** является маркером конца блока, используемым при перечислении переходов (см. далее 4.4). Символы угловых скобок и амперсанда необходимо заменять для совместимости с форматом GraphML (см. далее 4.3).

Примеры действий:

- `Сенсор.ПоискЦелиПоДистанции(мин)`
- `Timer1.restart`
- `LED1.blink(5, 1); LED1.off();`
- `restart`

Пример блока поведения 1:

```
Сенсор.ПоискЦелиПоДистанции(мин)
СпособностьОчистка.Активировать(мин)
```

Пример блока поведения 2:

```
МодульДвижения.ДвигатьсяКЦели
Счетчик.Увеличить(1)
```

Если это позволяет язык целевой платформы, допустимо писать несколько действий в одну строку, но стилистически это нежелательно.

В соответствии с НКФП.ПРИМС, в качестве собственных действий целевой платформы не рекомендуется использовать следующие ключевые слова: *propagate*, *block u defer*. Использование их в качестве одиночных действий (в частности одиночного *defer* кроме как маркера отложенного события) запрещено.

5.3 Ограждающее условие

Для ограничения и параметризации событий используются предикаты – выражения, принимающие при вычислении значение «истина» или «ложь». В общем случае это логические конструкции с использованием переменных, констант, операторов сравнения и логических операций.

Для обозначения ограждающего условия логическое выражение приводится в одну строку и оборачивается в квадратные скобки. Например,

```
[not is_target_dw()]
[Counter1.value + 5 == Counter2.value]
[else]
[Счетчик.ТекущееЗначениеСчетчика &gt;= 2]
```

Последнее условие будет истинным, если значение счётчика больше или равно 2.

Использовать единственное ключевое слово *else* в качестве ограждающего условия можно **исключительно** для выражения перехода при ложном значении всех остальных ограждающих условий (см. далее). В иных случаях применение одиночного *else* некорректно.

В большинстве случаев, для операторов используются обозначения из языка программирования Си, как наиболее распространённые в общемировой практике. Но стоит учитывать, что GraphML в основе является XML, и поэтому некоторые символы, как в примере выше, заменяются специальными обозначениями (мнемониками). Операторы с учётом этих особенностей приведены в таблице 1.

Таблица 1. Операторы условия

Оператор	Обозначение	В файле
И	&&	&&
ИЛИ	||	
НЕ	!	
Исключающее ИЛИ	^	
Больше, чем	>	>
Меньше, чем	<	<
Больше или равно	>=	>=
Меньше или равно	<=	<=
Равно	==	
Не равно	!=	

Использование квадратных скобок в условии допускается попарно (т.е. когда последовательно идут открывающая и закрывающая скобки) или при экранировании обратной косой чертой (\[).

5.4 Переход

Главная функциональная единица РИМС наряду с состоянием – это **переход**. Оно определяется прежде всего событием, являясь реакцией на него. На событие может накладываться ограждающее условие, и событие может сопровождаться выполнением поведения. Таким образом, описание перехода принимает следующий вид: на первой строке приводится событие с опциональным условием и косой чертой в конце строки (это маркер события), далее приводится многострочный блок поведения *без пустых строк*. Маркер конца блока поведения – **пустая строка**. Между событием, условием и косой чертой может быть любое число пробелов.

```
событие[условие]/
блок поведения
...
конец блока поведения
(пустая строка)
```

Например:

```
  entry/
  ↳
```

```
TIMER_1M/
update_power()
↳
Сенсор.ЦельПолучена[Счетчик.ТекущееЗначениеСчетчика &gt;= 2]/
МодульДвижения.ДвигатьсяКЦели(мин)
Таймер.ТаймерЗапуск(3)
```

Наличие условия опционально, как и поведения. Хотя отсутствие последнего во внутреннем переходе (т.е. внутри состояния) может показаться бесполезным, оно может применяться для перехвата передаваемых событий.

Согласно формату, во внешних переходах представлена только одна совокупность событий, условия и поведения. Содержимое состояния в свою очередь — это **набор внутренних переходов**, разделенных пустыми строками. Внутри состояния может быть любое число внутренних переходов, в том числе нулевое. Во внешнем переходе (как уже упоминалось) должно быть ровно один блок перехода. В случаях, допускаемых стандартом, возможен переход без события. Например, исходящий переход от начальной вершины. Большее число событий в переходе можно применять для объединения нескольких переходов с одним направлением. Аналогично, выше приведённый пример целиком является корректной цепочкой событий.

В составных переходах в качестве ограждающего условия допускается использовать единственное ключевое слово *else*, имеющее семантику, идентичную конструкции ветвления в императивных языках программирования. Ограждающие условие может содержать «else» только и только если существуют другие переходы из данного состояния по тому же набору событий, содержащие ограждающие условие без «else».

6 Стандартные расширения формата

В этом разделе приводятся концепции и функции, не включенные в ядро формата, но которые могут быть полезными разработчикам приложений, а в некоторых случаях могут перейти в ядро формата следующей версии. Эти расширения могут нести как функциональное назначение, а также расширять возможности по визуализации документа.

Используемые расширения определяются конкретным приложением и целевой платформой (например, наличие описания компонентов), равно как приложение и платформа могут вводить и применять собственные расширения над форматом для решения частных задач. Таким образом, стандартные расширения следует рассматривать как **рекомендованную реализацию** типовых концепций.

6.1 Динамические компоненты

Комментарий (тег *node* с ключом *dNote*, равным *informal*), который содержит заголовок (*dName*) *CGML_COMPONENT*, считается компонентом схемы. Это необходимо для целевых платформ, где состав компонентов не фиксирован и может меняться в зависимости от задачи.

В содержимом комментария (*dData*) после заголовка приводятся **параметры** компонента по той же схеме, что и метапараметры (набор строк вида *название/ значение*, разделённые пустой строкой). Список параметров и их опциональность определяется целевой платформой и/или редактором, но предлагается в качестве **обязательных** параметры *id*, определяющий **идентификатор** компонента и *type*, определяющий его **тип**.

```
<node id="cLED1">
  <data key="dNote">formal</data>
  <data key="dName">CGML_COMPONENT</data>
  <data key="dData">id/ LED1

type/ LED

name/ Светодиод

description/ Встроенный в плату светодиод, чтобы им мигать

pin/ 13
  </data>
</node>
```

Для удобства чтения метакомментарий компонента рекомендуется связывать с метакомментарием документа:

```
<edge id="coreMeta-cLED1" source="coreMeta" target="cLED1">
  <data key="dPivot"></data>
</edge>
```

6.2 Цветовая маркировка

Стандарт НКФП.ПРИМС задаёт … Реализация стиля зависит от редактора … Формат предлагает инструмент для смыслового выделения.

Для указания цвета элемента (обводка прямоугольного узла, цвет линии перехода или цвет обозначения вершины) рекомендуется использовать ключ *dColor*, указываемый соответственно в теге узла или перехода.

```
<data key="dColor">#444444</data>
```

В значении предлагается поддерживать веб-цвета с префиксом #, а также RGB и RGBA. Поддержка ключевых слов веб-цветов (*red*, *green* и т.д.) из CSS желательна.

6.3 Состояния со вложенной машиной состояний

Узел с ключом *dSubmachineState* указывает на то, что состояние содержит вложенную машину состояний. Содержимое ключа трактуется как путь ко **внешней машине состояний**, URL или идентификатор внутри документа:

```
<graph id="Main">
…
  <node id="externalSM">
    <data
key="dSubmachineState">file://../VacuumCleanAndFlush.graphml</data>
  </node>
  <node id="externalSM2">
    <data key="dSubmachineState">VacuumCharge</data>
  </node>
…
</graph>

<graph id="VacuumCharge">
  <data key="dStateMachine"/>
  <!-- ещё одна МС в пределах документа -->
  …
</graph>
```

При наличии в узле ключа *dSubmachineState*, использование других тегов недопустимо.

Для полноценной работы со вложенными машинами состояний необходимо использование точек входа и выхода (*entryPoint* и *exitPoint*), которые в данной версии формата заявлены как зарезервированные.

6.4 Форматированные комментарии

В неформальных комментариях, особенно описывающего характера, возникает потребность выделить участки текста или привести внешнюю ссылку. Для обозначения формата комментария предлагается ключ *dMarkup* (используемый только в связке с ключом *dNote*), содержимое которого содержит указание языка разметки.

Для форматирования текста комментариев рекомендуется язык разметки Markdown, являющийся одним из наиболее распространённых способов описания форматируемого текста, в том числе в программной документации и комментариях к коду. Языку Markdown соответствует ключ *dMarkup*, равный соответственно *markdown*.

```
<node id="noteX">
  <data key="dNote">informal</data>
  <data key="dMarkup">markdown</data>
  <data key="dData">И как мы опишем **вот это**?</data>
</node>
```

Аналогично, значение *plain* указывает на использование обычного текста (что делается по умолчанию). Кроме Markdown, разработчики редакторов могут добавлять поддержку других языков разметки, например HTML, вики-разметку и т.д.

Для сокращения повторных указаний ключа предлагается использование мета-параметра *markupLanguage*, устанавливающего значение языка разметки формальных комментариев.

6.5 Псевдосостояния истории

При поддержке этого целевой платформой, для обозначения псевдосостояния локальной истории используется ключ *dVertex*, равный *shallowHistory*, а для глубокой истории – *deepHistory*.

```
<node id="vHistory">
  <data key="dVertex">shallowHistory</data>
</node>
```

В подграфе на родительском уровне должно присутствовать не более **одного** псевдосостояния локальной истории и **одного** псевдосостояния глубокой истории. Далее к этим узлам как целевым приводится неограниченное число ребер-переходов

с событиями. Использование псевдосостояний истории в качестве источника перехода некорректно.

6.6 Начальная и конечная точки ребра

Для привязки ребра к конкретным местам исходного и целевого узлов предлагается использовать ключи *dSourcePoint* и *dTargetPoint*, в которых аналогично тегу *dGeometry* через тег *point* указывается координата **относительно** левого верхнего угла соответствующего узла (или относительно центра вершины):

```
<edge id="coreMeta-cLED1" source="coreMeta" target="cLED1">
  <data key="dPivot"/>
  <data key="dSourcePoint"><point x="0" y="50"/></data>
  <data key="dTargetPoint"><point x="-50" y="0"/></data>
</edge>
```

С помощью ключа *dTargetPoint* также можно указывать конкретную точку предмета комментирования при указании на элемент целиком (пустой *dPivot*).

6.7 Координаты метки ребра

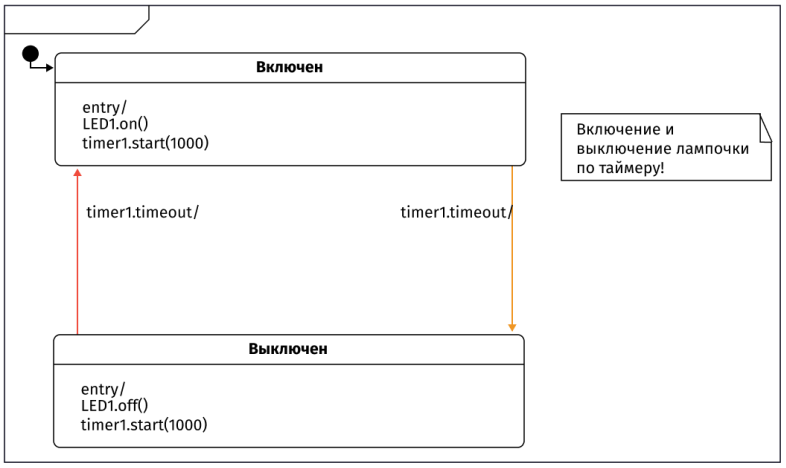
Для указания точных координат метки (label) ребра предлагается использовать ключ *dLabelGeometry*, в котором аналогично тегу *dGeometry* через тег *point* или *rect* указываются параметры метки. Тег *point* используется, если метка оформлена как текстовая строка, и указывает координаты её центра. Тег *rect* используется, если метка оформлена как прямоугольник, и описывает параметры этого прямоугольника. Координаты задаются **относительно** верхнего левого угла родительского для исходящей вершины узла:

```
<edge id="n3-n0::n1" source="n3" target="n0::n1">
  <data key="dData">Сенсор.ЦельПолучена/
  </data>
  <data key="dLabelGeometry">
    <rect x="200" y="400" width="200" height="100"/>
  </data>
</edge>
```

Приложение А. Примеры схем

А.1 Мигалка на Ардуино

Приведённая схема реализует программу, функционально идентичную шаблону «Blinker» для платы Arduino Uno (схема РИМС приведена на рисунке ниже, метаузлы скрыты).




```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">

<data key="gFormat">Cyberiada-GraphML-1.0</data>

<key id="gFormat" for="graphml" attr.name="format"
     attr.type="string"/>
<key id="dName" for="graph" attr.name="name" attr.type="string"/>
<key id="dName" for="node" attr.name="name" attr.type="string"/>
<key id="dStateMachine" for="graph" attr.name="stateMachine"
attr.type="string"/>
<key id="dSubmachineState" for="node" attr.name="submachineState"
attr.type="string"/>
<key id="dGeometry" for="graph" attr.name="geometry"/>
<key id="dGeometry" for="node" attr.name="geometry"/>
<key id="dGeometry" for="edge" attr.name="geometry"/>
<key id="dSourcePoint" for="edge" attr.name="sourcePoint"/>
<key id="dTargetPoint" for="edge" attr.name="targetPoint"/>
<key id="dLabelGeometry" for="edge" attr.name="labelGeometry"/>
<key id="dNote" for="node" attr.name="note" attr.type="string"/>
<key id="dVertex" for="node" attr.name="vertex" attr.type="string"/>
<key id="dData" for="node" attr.name="data" attr.type="string"/>
<key id="dData" for="edge" attr.name="data" attr.type="string"/>
<key id="dMarkup" for="node" attr.name="markup" attr.type="string"/>
```

```
<key id="dColor" for="node" attr.name="color" attr.type="string"/>
<key id="dColor" for="edge" attr.name="color" attr.type="string"/>
<key id="dPivot" for="edge" attr.name="pivot" attr.type="string"/>
<key id="dChunk" for="edge" attr.name="chunk" attr.type="string"/>

<graph id="G" edgedefault="directed">
    <data key="dStateMachine"/>
    <node id="coreMeta">
        <data key="dNote">formal</data>
        <data key="dName">CGML_META</data>
        <data key="dData">platform/ ArduinoUno

standardVersion/ 1.0

name/ Arduino Blinker

author/ Lapki IDE TEAM

description/ Включение и выключение лампочки по таймеру
        </data>
    </node>
    <node id="cLED1">
        <data key="dNote">formal</data>
        <data key="dName"> CGML_COMPONENT </data>
        <data key="dData">id/ LED1

type/ LED

name/ Светодиод

description/ Встроенный в плату светодиод, чтобы им мигать

pin/ 12
        </data>
    </node>
    <node id="ctimer1">
        <data key="dNote">formal</data>
        <data key="dName"> CGML_COMPONENT </data>
        <data key="dData">id/ timer1

type/ Timer

name/ Таймер

description/ Программный таймер.
        </data>
```

```
    </node>

    <node id="init">
        <data key="dVertex">initial</data>
        <data key="dGeometry">
            <point x="72" y="37"/>
        </data>
    </node>

   <node id="diod1">
        <data key="dName">Включен</data>
        <data key="dData">entry/
LED1.on()
timer1.start(1000)

        </data>
        <data key="dGeometry">
            <rect x="82" y="57" width="450.0" height="95"/>
        </data>
    </node>

    <node id="diod2">
        <data key="dName">Выключен</data>
        <data key="dData">entry/
LED1.off()
timer1.start(1000)

        </data>
        <data key="dGeometry">
            <rect x="81" y="334" width="450" height="95" />
        </data>
    </node>

    <node id="commentX">
        <data key="dNote"/>
        <data key="dData"> Включение и выключение лампочки по
таймеру! </data>
        <data key="dGeometry">
            <point x="640" y="114" />
        </data>
    </node>

    <edge id="init-edge" source="init" target="diod1"/>
    <edge id="edge0" source="coreMeta" target="ctimer1">
        <data key="dPivot"/>
    </edge>
```

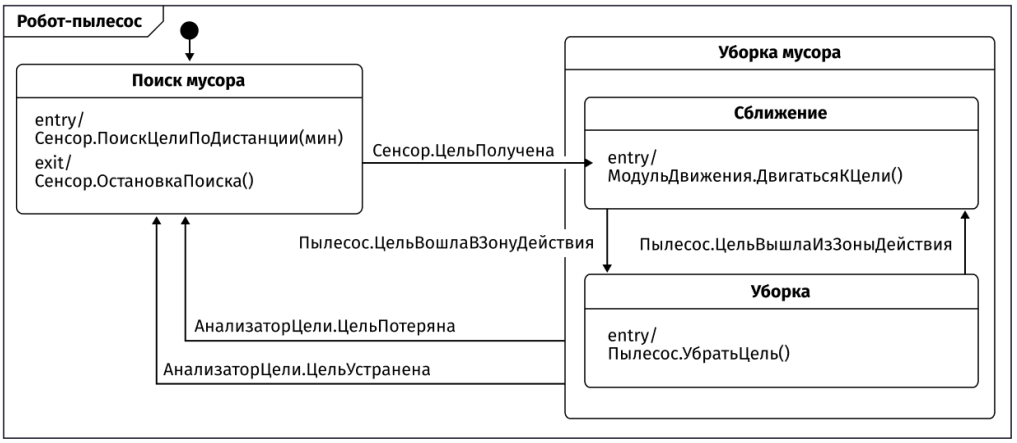
```
    <edge id="edge1" source="coreMeta" target="cLED1">
        <data key="dPivot"/>
    </edge>
    <edge id="edge3" source="diod1" target="diod2">
        <data key="dData">timer1.timeout/</data>
        <data key="dColor">#F29727</data>
        <data key="dLabelGeometry">
            <point x="457" y="173"/>
        </data>
    </edge>

    <edge id="edge4" source="diod2" target="diod1">
        <data key="dData">timer1.timeout/</data>
        <data key="dLabelGeometry">
            <point x="16" y="175"/>
        </data>
        <data key="dColor">#F24C3D</data>
    </edge>
</graph>

</graphml>
```

A.2 Автобортник

Приведённая схема является шаблоном поведения робота-пылесоса в игре «Академия» платформы Берлога (схема РИМС приведена на рисунке ниже, узел метаданных скрыт).




```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">

<data key="gFormat">Cyberiada-GraphML-1.0</data>

<key id="gFormat" for="graphml" attr.name="format"
     attr.type="string"/>
<key id="dName" for="graph" attr.name="name" attr.type="string"/>
<key id="dName" for="node" attr.name="name" attr.type="string"/>
<key id="dStateMachine" for="graph" attr.name="stateMachine"
attr.type="string"/>
<key id="dSubmachineState" for="node" attr.name="submachineState"
attr.type="string"/>
<key id="dGeometry" for="graph" attr.name="geometry"/>
<key id="dGeometry" for="node" attr.name="geometry"/>
<key id="dGeometry" for="edge" attr.name="geometry"/>
<key id="dSourcePoint" for="edge" attr.name="sourcePoint"/>
<key id="dTargetPoint" for="edge" attr.name="targetPoint"/>
<key id="dLabelGeometry" for="edge" attr.name="labelGeometry"/>
<key id="dNote" for="node" attr.name="note" attr.type="string"/>
<key id="dVertex" for="node" attr.name="vertex" attr.type="string"/>
<key id="dData" for="node" attr.name="data" attr.type="string"/>
<key id="dData" for="edge" attr.name="data" attr.type="string"/>
<key id="dMarkup" for="node" attr.name="markup" attr.type="string"/>
<key id="dColor" for="node" attr.name="color" attr.type="string"/>
<key id="dColor" for="edge" attr.name="color" attr.type="string"/>
<key id="dPivot" for="edge" attr.name="pivot" attr.type="string"/>
<key id="dChunk" for="edge" attr.name="chunk" attr.type="string"/>
```

```
<graph id="G" edgedefault="directed">
  <data key="dStateMachine"></data>

  <node id="nMeta">
    <data key="dNote">formal</data>
    <data key="dName">CGML_META</data>
    <data key="dData">platform/ BearsAcademyHoover

standardVersion/ 1.0

name/ Стандартный пылесос

author/ Матросов В.М.

contact/ matrosov@mail.ru

description/ Пример описания схемы,
который может быть многострочным, потому что так удобнее

target/ Hoover
    </data>
  </node>

  <node id="n0">
    <data key="dName">Уборка мусора</data>
    <data key="dData">entry/

exit/
</data>
    <data key="dGeometry">
      <rect x="-578.005" y="438.187256"
            width="672.532166" height="802.962646" /> </data>
    <graph id="n0::">
      <node id="n0::n1">
        <data key="dName">Сближение</data>
        <data key="dData">entry/
МодульДвижения.ДвигатьсяКЦели()

exit/
</data>
        <data key="dGeometry">
          <rect x="-525.738953" y="609.6686"
                width="468" height="170" /> </data>
      </node>
      <node id="n0::n2">
```

```
        <data key="dName">Уборка</data>
        <data key="dData">entry/
Пылесос.УбратьЦель()

exit/
</data>
        <data key="dGeometry">
          <rect x="-630.2711" y="206.705933"
                width="468" height="170" /> </data>
      </node>
    </graph>
  </node>
  <node id="n3">
    <data key="dName">Скан</data>
    <data key="dData">entry/
Сенсор.ПоискЦелиПоДистанции(мин)

exit/
Сенсор.ОстановкаПоиска()
</data>
    <data key="dGeometry">
      <rect x="-1582.03857" y="606.497559"
            width="468" height="330" /> </data>
  </node>
  <node id="init">
    <data key="dVertex">initial</data>
    <data key="dGeometry">
      <point x="-1472.03857" y="616.497559"/> </data>
  </node>

  <edge id="init-n3" source="init" target="n3"/>
  <edge id="n0-n3" source="n0" target="n3">
    <data key="dData">АнализаторЦели.ЦельУстранена/
</data>
  </edge>
  <edge id="n0-n3" source="n0" target="n3">
    <data key="dData">АнализаторЦели.ЦельПотеряна/
</data>
  </edge>
  <edge id="n3-n0::n1" source="n3" target="n0::n1">
    <data key="dData">Сенсор.ЦельПолучена/
</data>
  </edge>
  <edge id="n0::n1-n0::n2" source="n0::n1" target="n0::n2">
      <data key="dData">ОружиеЦелевое.ЦельВошлаВЗонуДействия/
</data>
```

```
  </edge>
  <edge id="n0::n2-n0::n1" source="n0::n2" target="n0::n1">
      <data key="dData">ОружиеЦелевое.ЦельВышлаИзЗоныДействия/
</data>
  </edge>

</graph>
</graphml>
```

A.3 Минимально допустимый документ

Приведённый документ реализует пустую машину состояний без платформы.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">

<data key="gFormat">Cyberiada-GraphML-1.0</data>

<key id="gFormat" for="graphml" attr.name="format"
     attr.type="string"/>
<key id="dName" for="graph" attr.name="name" attr.type="string"/>
<key id="dName" for="node" attr.name="name" attr.type="string"/>
<key id="dStateMachine" for="graph" attr.name="stateMachine"
attr.type="string"/>
<key id="dSubmachineState" for="node" attr.name="submachineState"
attr.type="string"/>
<key id="dGeometry" for="graph" attr.name="geometry"/>
<key id="dGeometry" for="node" attr.name="geometry"/>
<key id="dGeometry" for="edge" attr.name="geometry"/>
<key id="dSourcePoint" for="edge" attr.name="sourcePoint"/>
<key id="dTargetPoint" for="edge" attr.name="targetPoint"/>
<key id="dLabelGeometry" for="edge" attr.name="labelGeometry"/>
<key id="dNote" for="node" attr.name="note" attr.type="string"/>
<key id="dVertex" for="node" attr.name="vertex" attr.type="string"/>
<key id="dData" for="node" attr.name="data" attr.type="string"/>
<key id="dData" for="edge" attr.name="data" attr.type="string"/>
<key id="dMarkup" for="node" attr.name="markup" attr.type="string"/>
<key id="dColor" for="node" attr.name="color" attr.type="string"/>
<key id="dColor" for="edge" attr.name="color" attr.type="string"/>
<key id="dPivot" for="edge" attr.name="pivot" attr.type="string"/>
<key id="dChunk" for="edge" attr.name="chunk" attr.type="string"/>

<graph id="G" edgedefault="directed">
    <data key="dStateMachine"/>
    <node id="coreMeta">
        <data key="dNote">formal</data>
        <data key="dName">CGML_META</data>
        <data key="dData">standardVersion/ 1.0
        </data>
    </node>
</graph>

</graphml>
```

Приложение Б. Перечень объявлений ключей

В этом приложении приводятся объявления всех ключей, описанных в формате (включая стандартные расширения). Этот блок можно брать и вставлять в документ, убирая по необходимости неиспользуемые ключи.

```
<key id="gFormat" for="graphml" attr.name="format"
     attr.type="string"/>
<key id="dName" for="graph" attr.name="name" attr.type="string"/>
<key id="dName" for="node" attr.name="name" attr.type="string"/>
<key id="dStateMachine" for="graph" attr.name="stateMachine"
attr.type="string"/>
<key id="dSubmachineState" for="node" attr.name="submachineState"
attr.type="string"/>
<key id="dGeometry" for="graph" attr.name="geometry"/>
<key id="dGeometry" for="node" attr.name="geometry"/>
<key id="dGeometry" for="edge" attr.name="geometry"/>
<key id="dSourcePoint" for="edge" attr.name="sourcePoint"/>
<key id="dTargetPoint" for="edge" attr.name="targetPoint"/>
<key id="dLabelGeometry" for="edge" attr.name="labelGeometry"/>
<key id="dNote" for="node" attr.name="note" attr.type="string"/>
<key id="dVertex" for="node" attr.name="vertex" attr.type="string"/>
<key id="dData" for="node" attr.name="data" attr.type="string"/>
<key id="dData" for="edge" attr.name="data" attr.type="string"/>
<key id="dMarkup" for="node" attr.name="markup" attr.type="string"/>
<key id="dColor" for="node" attr.name="color" attr.type="string"/>
<key id="dColor" for="edge" attr.name="color" attr.type="string"/>
<key id="dPivot" for="edge" attr.name="pivot" attr.type="string"/>
<key id="dChunk" for="edge" attr.name="chunk" attr.type="string"/>
```

Приложение B. Дерево тегов

В этом приложении приводится структура тегов, используемая в CGML. Так как формат во многом опирается на теги data, для информативности эти теги в дереве будут указываться с ключом, например, data<dData>. Многоточие указывает на рекурсивность (т.е. структура тега повторяет ту, что указана для этого тега выше).

```
graphml
  data<gFormat>
  key // описание ключей data
  graph
    data<dName>
    data<dGeometry>
      point
      rect
    node
      data<dNote>
      data<dSubmachineState>
      data<dName>
      data<dGeometry> ...
      data<dColor>
      data<dVertex>
      data<dMarkup>
      data<dData>
      graph* ...
    edge
      data<dPivot>
      data<dChunk>
      data<dData>
      data<dGeometry> ...
      data<dColor>
      data<dSourcePoint>
        point
      data<dTargetPoint>
        point
      data<dLabelGeometry>
        point
        rect
```