**Руководство для разработчика**

Компонент: Eco.FSM1

Статус: Черновик

Дата: Ноябрь 8, 2020

Версия: 1.0

|  |  |
| --- | --- |
| **Авторы** | **Компания** |
| Владимир Башев | ПИРФ |
|  |  |

Содержание

[**1.** **Обзор** 5](#_Toc184301708)

[**1.1.** **Введение** 5](#_Toc184301709)

[**1.2.** **Классификация автоматов** 6](#_Toc184301710)

[**1.3.** **Примечание** 6](#_Toc184301711)

[**1.4.** **Ссылки** 6](#_Toc184301712)

[**1.5.** **Введение** 6](#_Toc184301713)

[**2.** **Теория и практика** 7](#_Toc184301714)

[**2.1.** **Конечный автомат** 8](#_Toc184301715)

[**2.2.** **Преобразование алфавитной информации** 9](#_Toc184301716)

[**2.3.** **Элементарные и регулярные события** 16](#_Toc184301717)

[**2.4.** **Состояния и переходы** 16](#_Toc184301718)

[**2.5.** **Организация управления автоматом** 18](#_Toc184301719)

[**3.** **Компонент Eco.FSM1** 20](#_Toc184301720)

[**4.** **Интерфейс IEcoFSM1** 21](#_Toc184301721)

[**4.1.** **IEcoFSM1 описание на ECO IDL** 21](#_Toc184301722)

[**4.1.1.** **Функция CreateAlphabet** 21](#_Toc184301723)

[**4.1.2.** **Функция CreateAutomata** 21](#_Toc184301724)

[**4.1.3.** **Функция CreateEventProcessing** 21](#_Toc184301725)

[**5.** **Интерфейс IEcoFL1Alphabet** 23](#_Toc184301726)

[**5.1.** **IEcoFL1Alphabet описание на ECO IDL** 23](#_Toc184301727)

[**5.1.1.** **Функция get\_Name** 23](#_Toc184301728)

[**5.1.2.** **Функция get\_Id** 23](#_Toc184301729)

[**5.1.3.** **Функция AddSymbolSet** 23](#_Toc184301730)

[**6.** **Интерфейс IEcoFSM1Automata** 25](#_Toc184301731)

[**6.1.** **IEcoFSM1Automata описание на ECO IDL** 25](#_Toc184301732)

[**6.1.1.** **Функция get\_Name** 25](#_Toc184301733)

[**6.1.2.** **Функция get\_Id** 25](#_Toc184301734)

[**6.1.3.** **Функция AddState** 26](#_Toc184301735)

[**6.1.4.** **Функция get\_States** 26](#_Toc184301736)

[**6.1.5.** **Функция AddEvent** 26](#_Toc184301737)

[**6.1.6.** **Функция get\_Events** 26](#_Toc184301738)

[**6.1.7.** **Функция AddTransition** 26](#_Toc184301739)

[**6.1.8.** **Функция get\_Transitions** 26](#_Toc184301740)

[**6.1.9.** **Функция AddTransitionById** 27](#_Toc184301741)

[**7.** **Интерфейс IEcoFSM1State** 28](#_Toc184301742)

[**7.1.** **IEcoFSM1State описание на ECO IDL** 28](#_Toc184301743)

[**7.1.1.** **Функция get\_Id** 28](#_Toc184301744)

[**7.1.2.** **Функция get\_Name** 28](#_Toc184301745)

[**7.1.3.** **Функция set\_Name** 29](#_Toc184301746)

[**7.1.4.** **Функция IsInitial** 29](#_Toc184301747)

[**7.1.5.** **Функция set\_Initial** 29](#_Toc184301748)

[**7.1.6.** **Функция IsFinal** 29](#_Toc184301749)

[**7.1.7.** **Функция set\_Final** 29](#_Toc184301750)

[**7.1.8.** **Функция IsTrap** 29](#_Toc184301751)

[**7.1.9.** **Функция set\_Trap** 29](#_Toc184301752)

[**7.1.10.** **Функция get\_Parameter** 30](#_Toc184301753)

[**7.1.11.** **Функция set\_Parameter** 30](#_Toc184301754)

[**8.** **Интерфейс IEcoFSM1Event** 31](#_Toc184301755)

[**8.1.** **IEcoFSM1Event описание на ECO IDL** 31](#_Toc184301756)

[**8.1.1.** **Функция get\_Name** 31](#_Toc184301757)

[**8.1.2.** **Функция get\_Id** 31](#_Toc184301758)

[**8.1.3.** **Функция IsNull** 31](#_Toc184301759)

[**8.1.4.** **Функция set\_Null** 32](#_Toc184301760)

[**8.1.5.** **Функция get\_Parameter** 32](#_Toc184301761)

[**8.1.6.** **Функция set\_Parameter** 32](#_Toc184301762)

[**9.** **Интерфейс IEcoFSM1Transition** 33](#_Toc184301763)

[**9.1.** **IEcoFSM1Transition описание на ECO IDL** 33](#_Toc184301764)

[**9.1.1.** **Функция get\_Name** 33](#_Toc184301765)

[**9.1.2.** **Функция set\_Name** 33](#_Toc184301766)

[**9.1.3.** **Функция get\_Id** 33](#_Toc184301767)

[**9.1.4.** **Функция get\_Parameter** 34](#_Toc184301768)

[**9.1.5.** **Функция set\_Parameter** 34](#_Toc184301769)

[**9.1.6.** **Функция get\_Event** 34](#_Toc184301770)

[**9.1.7.** **Функция get\_Source** 34](#_Toc184301771)

[**9.1.8.** **Функция get\_Target** 34](#_Toc184301772)

[**10.** **Интерфейс IEcoFSM1EventProcessing** 35](#_Toc184301773)

[**10.1.** **IEcoFSM1EventProcessing описание на ECO IDL** 35](#_Toc184301774)

[**10.1.1.** **Функция get\_StateMachine** 35](#_Toc184301775)

[**10.1.2.** **Функция ResetStateMachine** 35](#_Toc184301776)

[**10.1.3.** **Функция get\_CurrentState** 36](#_Toc184301777)

[**10.1.4.** **Функция isInitialState** 36](#_Toc184301778)

[**10.1.5.** **Функция isFinalState** 36](#_Toc184301779)

[**10.1.6.** **Функция InvokeByEvent** 36](#_Toc184301780)

[**10.1.7.** **Функция InvokeById** 36](#_Toc184301781)

[**10.1.8.** **Функция InvokeByName** 36](#_Toc184301782)

[**11.** **В разработке …** 37](#_Toc184301783)

[**Приложение А** 37](#_Toc184301784)

1. **Обзор**

Данный документ описывает требования к реализации компонента Eco.FSM1 (Конечный Автомат).

* 1. **Введение**

Для решения задач по автоматизации программирования связанных с переводом языков высокого уровня на машинный язык необходимы наборы программных инструментов, одним из которых является конечный автомат без выхода (распознаватель). В компонентной среде Eco.Framework конечный автомат без выхода реализован в виде компонента Eco.FSM1, наименование которого имеет следующую расшифровку:

Eco – кодовое слово библиотеки

FSM – сокращение термина Finite State Machine (Конечный Автомат)

1 – основная версия интерфейса компонента

Ниже приводится таблица соответствие компонента согласно классификатору автоматов.

|  |  |
| --- | --- |
| Классификатор автоматов | Соответствие |
| **По типу модели** |  |
| * Абстрактные | ✓ |
| * Структурные |  |
| **По типу автомата:** |  |
| * Машина Тьюринга |  |
| * Линейно ограниченный автомат   (недетерминированная одноленточная машина Тьюринга) |  |
| * Автомат с магазинной памятью |  |
| * Конечный автомат | ✓ |
|  |  |
| **По видам записывающего устройства** |  |
| * Распознаватель – автомат без выхода | ✓ |
| * Преобразователь – автомат с выходом |  |
| **По видам считывающего устройства** |  |
| * Односторонние | ✓ |
| * Двусторонние |  |
| **По видам управляющего устройства** |  |
| * Детерминированные | ✓ |
| * Недетерминированные |  |
| **По видам внешней памяти** |  |
| * Без внешней памяти | ✓ |
| * С ограниченной внешней памятью |  |
| * С неограниченной внешней памятью |  |
|  |  |

Ввиду сложности и многообразия подходов изучения связанных с теорией автоматов, а также используемой терминологии, следующий пункт призван помочь разобраться с классификаций автоматов.

* 1. **Классификация автоматов**

По типу модели:

Абстрактный автомат - это математическая модель дискретного устройства, имеющего один входной канал, куда поступают последовательности символов какого-либо языка, и один выходной канал, с которого снимают последовательности символов какого-либо другого языка

* 1. **Примечание**
* Ключевые слова в документе
  1. **Ссылки**

Данный параграф содержит ссылки на информацию, помогающую понять данный документ:

[] – наименование ссылки

Доступен по: <http://адрес>

* 1. **Введение**

Описание.

1. **Теория и практика**

При изучении теории формальных языков и абстрактной теории автоматов появляется закономерный вопрос: - «Как отобразить теорию на практике для решения задач?».



Другой не маловажной проблемой на практике при автоматном подходе разработки ПО, является выбор уровня абстракции элементарных операций при моделировании сущности со сложным поведением.



В этом параграфе даются пояснения по отображению понятий изложенных в теории формальных языков и абстрактной теории автоматов на их практическую реализацию. Описываемое далее решение не претендует на лучшее по эффективности, но тем не менее является более универсальным и старается учитывать тенденции технологического развития ЭВМ используя при этом ACOM технологию как программную составляющую.

* 1. **Конечный автомат**

Рассматриваемое решение - ***конечный автомат без выхода*** (теория автоматов), является частным случаем распознавателя для регулярных языков (теория формальных языков), где головка движется всегда вправо на каждом такте работы и отсутствует внешняя память.



Рис. 1 Конечный автомат Рис. 2 Временная диаграмма работы автомата

Из теории определение конечного автомата без выхода выглядит следующим образом:

***A*** – {*Q, Σ, δ, q0, F*} , где

*Q* – конечное множество состояний автомата

*Σ* - конечное множество допустимых входных символов

*δ* - функция переходов вида *δ*(*q, a*) = *qk*, *a* ∈ *Σ*, *q* ∈ *Q*, *qk* ⊆ *Q*

*q*0 - начальное состояние автомата *q*0 ∈ *Q*

*F* - непустое множество конечных состояний автомата *F*⊆ *Q*

Результатом работы конечного автомата без выхода является конечное слово α ∈ *Σ \**, которое или принимается (допуск) или отвергается (отказ).

Другими словами конечный автомат определяет некоторое подмножество *L*(***A***) ⊆ *Σ \** слов, которое он принимает, где

*L*(***A***) – язык, принимаемый (или допускаемый) конечным автоматом без выхода ***A***

Конечный автомат удобнее представить в виде ***направленного графа с весами***, где узлы являются состояниями, ребра - это переходы, а весы входной символ или набор символов отображающие событие.



Отталкиваясь от теоретического представления, на практике, для решения задач распознавания регулярных языков, необходимо построить распознаватель (конечный автомат без выхода) на основе имеющейся грамматики языка программирования. За построение конечного автомата отвечает интерфейс ***IEcoFSM1*** компонента **Eco.FSM1**, который имеет три метода:

**CreateAlphabet** – создание экземпляра алфавита описывающее конечное множество допустимых входных символов, для работы с которым возвращается интерфейс ***IEcoFL1Alphabet***

**CreateAutomata** – создание экземпляра автомата описывающее конечное множество состояний и функций переходов с использованием входного алфавита, для работы с которым возвращается интерфейс ***IEcoFSM1Automata***

**CreateEventProcessing** – создание экземпляра обработки событий (генерации импульсов) описывающее изменение состояний автомата во времени по входному символу (событию), для работы с которым возвращается интерфейс ***IEcoFSM1EventProcessing***

Далее необходимо рассмотреть формирование алфавита.

* 1. **Преобразование алфавитной информации**

Данный пункт дает пояснение назначения использования интерфейсов ***IEcoFL1Alphabet*** и ***IEcoFL1SymbolSet*** в работе конечного автомата КА

Для решения прикладных задач, связанных с формальными языками, возникает необходимость в способе задания исходной информации. Не маловажным вопросом остается: - «*Как это сделать?*». В большинстве электронно-вычислительных машин (ЭВМ) для преобразования информации используется двоичная система счисления, где числа записываются с помощью двух символов (0 и 1). Другими словами, для задания исходной информации используется стандартный двухбуквенный, или двоичный, алфавит, а две его буквы обозначаются нулем и единицей. В качестве минимальной единице измерения информации, которое принимает два значения (0 и 1), используется **бит** (*англ*. **bit**). Также необходимо отметить о минимальной единице хранения и обработки цифровой информации, которой является **байт** (*англ*. **byte**). В современных ЭВМ байт состоит из 8 бит и, соответственно, может принимать одно из 256 (от 0 до 255) различных значений (состояний, кодов), к которому можно адресоваться (см. Рис. 1 и Рис. 2 ).



Рис. 3 Байт



Рис. 4 Адресация

Исходя из данного представления, возникает еще больше вопросов в способе задания исходной информации: - «Какой используется порядок следования данных?», «Какая используется последовательность передачи данных?»

В ячейке памяти размером 1 Байт показанной на Рис. 2 , биты хранятся в прямом порядке (*англ.* **big endian**), от старшего к младшему. Для представления исходной информации, требующей больше 256-ти состояний, т.е. больше чем 1 Байт, необходимо использовать 2 Байта. Если количество состояний необходимое для представления исходной информации превышает возможное, к примеру 2 Байта, то необходимо увеличить длину еще на один Байт.



Рис. 5 Порядок следования

Таким образом хранить и передавать данные необходимо в прямом порядке. Стоит отметить, что в памяти ЭВМ, по младшему адресу ячейки будет храниться старший байт, а по адресу ячейки + 1, будет храниться младший байт для кодируемой информации длиной два байта.



Рис. 6 Порядок хранения

Ответив на вопросы порядков следования и хранения данных в современных ЭВМ, отобразим абстрактный автомат на структуру данных ЭВМ



Рис. 7 Отображение абстрактного автомата

Как видно из Рис, отображение будет хорошо работать при условии: - если для представления входного символа достаточно 8-ми бит информации, и перемещение входной головки автомата эквивалентно смещению указателя на один байт. А что, если для представления входного символа достаточно всего 4-х бит, т.е. в одной ячейке памяти может расположиться 2 входных символа. Или для входного символа достаточно всего одного бита информации, тогда в ячейке памяти может расположиться 8-мь входных символов (двоичный алфавит). Исходя из этих примеров представления входного символа, необходим механизм перемещения входной головки внутри байта.



Вторая проблема заключается в следующем: допустим для представления входного символа требуется 7 бит информации, а на входной ленте, символы выравнены по границе одного байта (8 бит), т.е. самый старший 7-й бит является незначащим и может принимать любое значение (0 или 1). Другой пример, где для представления водного символа требуется 5-ть бит информации и оставшиеся три бита информации являются незначащими.



Третья проблема с перемещением входной головки возникает, когда для представления входного символа требуется больше чем 8 бит информации, т.е. входной символ может быть представлен 12-тью, 16-тью, 24-ми, 32-мя или 56-тью битами информации.



Чтобы глубже понять проблематику, обратимся к абстрактной теории автоматов и рассмотрим примеры из книги «Синтез цифровых автоматов» В.М. Глушков, Москва 1962:

«*Пример 3. Исходная информация – произвольное слово в русском алфавите. Преобразование информации состоит в замене каждой буквы алфавита порядковым номером, записанным в десятичной системе счисления. При этом для соблюдения условия равенства длин комбинаций букв нового алфавита, представляющие буквы старого алфавита, необходимо первую букву обозначить комбинацией 01, вторую –комбинацией 02 и т.д. В результате такого преобразования слово «дом» преобразуется в слово «051412», слово «бкзд» - в слово «02100805» и т.д.*

*Пример 4. Исходная информация – произвольное целое десятичное число (слово в алфавите, состоящем из десяти цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9); преобразование информации состоит в замене каждой четной цифры нулем, а каждой нечетной цифры – единицей. Слово «125» при этом преобразуется в слово «101», слово «0342» - в слово «0100» и т.д.*»

Из приведенных примеров видно, что преобразование для примера 3 является простейшим и эквивалентным, в противоположность примеру 4 в котором преобразование простейшее, но не эквивалентное.

С абстрактной точки зрения ***преобразование информации*** – это отображение одного класса явлений в другой класс явлений.

Проблема преобразования информации с использованием стандартного двоичного алфавита используемого современными ЭВМ «*состоит в получении из одного ряда нулей и единиц другого ряда нулей и единиц, отличающихся от первого либо числом символов, либо их взаимным расположении, либо тем и другим вместе*» (В.М. Глушков).

Данную проблему при рассмотрении регулярных языков по классификации Хомского позволяют решить конечные автоматы - КА (*англ.* **Finite-Automata - FA**)

Стоит отметить что преобразованием информации выполнятся конечным автоматом с выходом (преобразователем), в нашем случае рассматривается конечный автомат без выхода (распознаватель), для которого достаточно распознать слово в алфавите. Но в обоих типах автоматов с выходом и без выхода, задается алфавит входных символов, причем из примеров 3 и 4 видно, что для разных задач длина букв может отличаться.

Таким образом для работы конечного автомата необходим способ задания исходной информации (ряда нулей и единиц), формирования алфавита входных символов с условием равенства их длин и использованием стандартного двоичного алфавита.

Также стоит отметить, что для целого ряда задач в формирование алфавита входных символов используются готовые стандартизованные символы, к ним можно отнести кодировки ASCII, Unicode (UTF-8, UTF-16, UTF-32). К примеру:

|  |  |  |  |
| --- | --- | --- | --- |
| Кодировка | Латинская строчная буква «a» | Длина в битах | Длина в байтах |
| ASCII | 1100001 | 7 | 1 |
| UTF-8 | 01100001 | 8 | 1 |
| UTF-16 | 00000000, 01100001 | 16 | 2 |
| UTF-32 | 00000000, 00000000, 00000000, 01100001 | 32 | 4 |

Если для решения предполагаемой задачи требуется использовать, только строчные буквы русского языка, то достаточно каждой из 32 букв русского алфавита сопоставить одно из 32 =2 5 возможных слов длины 5 в двоичном алфавите. К примеру: букве «а» сопоставлен код 0000, букве «б» – код 00001 и т.д. Для наглядности, приведем сопостановление русской строчной буквы «а» с использованием стандартных кодировок:

|  |  |  |  |
| --- | --- | --- | --- |
| Кодировка | Русская строчная буква «a» | Длина в битах | Длина в байтах |
| - | 00000 | 5 | 1 |
| ISO 8859-5 | 11010000 | 8 | 1 |
| UTF-8 | 11010000, 10110000 | 11 | 2 |
| UTF-16 | 00000100, 00110000 | 16 | 2 |
| UTF-32 | 00000000, 00000000, 00000100, 00110000 | 32 | 4 |

Получается, что задание исходной информации, само по себе является задачей, требующей определенное решение в виде некой абстракции для описания алфавита символов.

Для описания алфавита символов, будем использовать интерфейсы ***IEcoFL1Alphabet*** и ***IEcoFL1SymbolSet*** по работе с объектом алфавита формальных языков.

Следует отметить, что в ряде случаев задания исходной информации, нет необходимости посимвольного сопоставления, достаточно задать диапазон или диапазоны сопоставления символов.

С алфавитом входных символов тесно связаны такие понятия как элементарные и регулярные события.

* 1. **Элементарные и регулярные события**

Данный пункт дает пояснение назначения использования интерфейсов ***IEcoFSM1Event*** и ***IEcoFSM2RegularEvent*** в работе конечного автомата КА

В работе конечного автомата по преобразованию информации используются понятия элементарные события и регулярные события. Определение данных понятий возьмем из книги «Синтез цифровых автоматов» В.М. Глушков, Москва 1962:

«*Элементарными событиями в алфавите Σ=(x1, x2, …, xn) называются n+1 одноэлементных событий x1, x2, …, xn, ε, где ε – пустое слово. Любое событие, которое можно получить из элементарных событий в результате применения конечного числа раз дизъюнкции, умножения и итерации, называется регулярным событием, а всякое его представление через элементарные события и указание операции – регулярным выражением*»

Из определения видно, что события представлены набором символов (букв) алфавита, следовательно, при описании событий, целесообразно использовать интерфейс набора символов алфавита ***IEcoFL1SymbolSet***

🕮 **Примечание:** *Интерфейс по работе с регулярными событиями не входит в реализацию компонента конечного автомата первой версии.*

* 1. **Состояния и переходы**

Данный пункт дает пояснение назначения использования интерфейсов ***IEcoFSM1State*** и ***IEcoFSM1Transition*** в работе конечного автомата КА

Основным элементом конечного автомата является устройство управления, которое состоит из регистра состояния и таблицы переходов (эквивалент логике переходов, *англ.* Transition Logic). Таблица переходов описываются правила перехода между состояниями в зависимости от текущего состояния и входных событий, которое программно может быть реализовано по-разному, как пример, в виде условных операторов (эквивалент логических вентилей, *англ.* Logic Gates) или таблицы в памяти (эквивалент логических ячеек памяти, *англ.* ROM-based)

В первую очередь данный компонент является гибким и универсальным инструментом проектирование конечного автомата используя интерфейсы для описания состояний, переходов и событий в виде объектов, что в свою очередь позволяет управлять автоматом используя логику условных операторов. Во вторую очередь на базе созданных объектов конечного автомата (алфавита, состояний, переходов, событий) создавать эффективную таблицу переходов в ячейках памяти для управления автоматом, таким образом программно реализуя оба подхода (условные операторы и таблица в памяти).

По аналогии с входным символом для организации хранения таблицы переходов в ячейках памяти встает вопрос, связанный сколько необходимо бит информации для хранения одного состояния в зависимости от количества состояний и количества входных символов.

На рис отображена организация памяти таблицы переходов для символов длиной в 8 бит и трех состояний. Для состояния нам необходимо всего два бита информации, но с учетом организации памяти в ЭВМ, с минимально адресуемой единицей информации в 1 байт, вынуждены использовать ячейку памяти целиком под одно состояние, таким образом для таблицы переходов нужно 768 ячеек памяти (байт).





Теперь рассмотрим ситуацию, в которой таблица переходов содержит 368 состояний, где для состояния нам нужно 9 бит информации, соответственно под одно состояние нам необходимо 2 байта информации, таким образом для таблицы переходов нужно: 256 входных символов \* 368 состояний \* 2 ячейки памяти = 188 416 ячеек памяти (байт).

Рассмотрим еще одну ситуации в которой длина символа составляет 16 бит при тех же 368 состояниях, при заданных параметрах для таблицы переходов нужно: 65 436 входных символов \* 368 состояний \* 2 ячейки памяти = 48 234 496 ячеек памяти (байт).

Как видно из расчетов приведенных примеров, получаемые таблицы переходов, требуют больших ресурсов (объемов памяти). В большинстве решаемых задач, связанных с распознаванием входных слов, зачастую для больших групп символов в конечном автомате определены одинаковые переходы из одного состояния в другое. Для решения данных проблем используют различные приемы программирования и применяют алгоритмы минимизации конечных автоматов. В частности, при описании грамматики регулярными выражениями, для набора символов используется понятие символьный класс.

Использование символьных классов позволяет в значительной степени сократить объем памяти при построении конечных автоматов. Но процесс сопоставления символа к классу символов, по своей сути является простейшим автоматом. Таким образом при использовании символьных наборов, мы получаем вложенный автомат



Возможность использования символьных наборов позволяет строить автоматы, как с со постановлением к символьному классу, так и без со постановления.





В виде графа:



* 1. **Организация управления автоматом**

Данный пункт дает пояснение назначения использования интерфейса ***IEcoFSM1EventProcessing*** в работе конечного автомата КА

В программной реализации конечного автомата, помимо описания используемого алфавита входных символов, событий, переходов и состояний автомата также необходимы механизмы управления автоматом во времени.

1. **Компонент Eco.FSM1**

Компонент, имеет следующие описание:

1. **Интерфейс IEcoFSM1**

Интерфейс ***IEcoFSM1*** отвечает за создание объектов для описания алфавита, состояний, событий и переходов конечного автомата, а также обработчика событий конечного автомата.

* 1. **IEcoFSM1 описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.idl" | | | |
| [  object,  uguid(7ACB8256-EEB3-40FA-8E83-9A9AF03464D5),  ] | | | |
| interface IEcoFSM1 : IEcoUnknown { | | | |
|  | | | |
| IEcoFL1Alphabet\* | | ***CreateAlphabet*** | ([in] char\_t\* Name); |
|  | | |  |
| IEcoFSM1Automata\* | | ***CreateAutomata*** | ([in] char\_t\* Name, |
|  | |  | [in] IEcoFL1Alphabet\* Input); |
|  | | |  |
| IEcoFSM1EventProcessing\* | | ***CreateEventProcessing*** | ([in] IEcoFSM1Automata\* Automata); |
|  | | | |
|  | | | |
| } | |  |  |

* + 1. **Функция CreateAlphabet**

Функция создает экземпляр алфавита формального языка с заданным именем ***Name*** и возвращает указатель на интерфейс алфавита ***IEcoFSM1Alphabet\*.***

* + 1. **Функция CreateAutomata**

Функция создает экземпляр конечного автомата с заданным именем ***Name***, указателем на интерфейс алфавита ***IEcoFL1Alphabet\**** и возвращает указатель на интерфейс конечного автомата ***IEcoFSM1Automata\*.***

* + 1. **Функция CreateEventProcessing**

Функция создает обработчик событий для конечного автомата, переданного как входной параметр ***Automata*** и возвращает указатель на интерфейс обработчика событий конечного автомата ***IEcoFSM1EventProcessing\*.***

1. **Интерфейс IEcoFL1Alphabet**
   1. **IEcoFL1Alphabet описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(34832ECF-CA8E-4B16-93F0-C735F0CA9F8D),  ] | | | |
| interface IEcoFL1Alphabet : IEcoUnknown { | | | |
|  | | | |
| char\_t\* | | ***get\_Name*** | ([in] void); |
|  | | |  |
| int16\_t | | ***get\_Id*** | ([in] void); |
|  | | | |
| IEcoFL1SymbolSet\* | | ***AddSymbolSet*** | ([in] char\_t\* Name); |
|  | | | |
| IEcoFL1SymbolSet\* | | ***get\_SymbolSet*** | ([in] int32\_t index); |
|  | |  |  |
| IEcoList1\* | | ***get\_SymbolSets*** | ([in] void); |
|  | |  |  |
|  | | | |
| } | |  |  |

* + 1. **Функция get\_Name**

Функция возвращает имя алфавита***.***

* + 1. **Функция get\_Id**

Функция возвращает идентификатор алфавита***.***

* + 1. **Функция AddSymbolSet**

Функция добавляет набор символов в алфавит с заданным именем ***Name*** и возвращает указатель на интерфейс ***IEcoFL1SymbolSet\*.***

1. **Интерфейс IEcoFSM1Automata**
   1. **IEcoFSM1Automata описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(6B73BD3F-22F6-462E-AC29-F5C1214E775E),  ] | | | |
| interface IEcoFSM1Automata : IEcoUnknown { | | | |
|  | | | |
| char\_t\* | | ***get\_Name*** | ([in] void); |
|  | | |  |
| int16\_t | | ***get\_Id*** | ([in] void); |
|  | | | |
| IEcoFSM1State\* | | ***AddState*** | ([in] char\_t\* Name); |
|  | | | |
| IEcoList1\* | | ***get\_States*** | ([in] void); |
|  | |  |  |
| IEcoFSM1Event\* | | ***AddEvent*** | ([in] char\_t\* Name, |
|  | |  | [in] int16\_t Id, |
|  | |  | [in] bool\_t isNull); |
|  | | | |
| IEcoList1\* | | ***get\_Events*** | ([in] void); |
|  | |  |  |
| IEcoFSM1Transition\* | | ***AddTransition*** | ([in] IEcoFSM1Event\* Event, |
|  | |  | [in] IEcoFSM1State\* Source, |
|  | |  | [in] IEcoFSM1State\* Target); |
|  | | | |
| IEcoList1\* | | ***get\_Transitions*** | ([in] void); |
|  | |  |  |
| IEcoFSM1Transition\* | | ***AddTransitionById*** | ([in] int16 EventId, |
|  | |  | [in] int16 SourceStateId, |
|  | |  | [in] int16 TargetStateId); |
|  | | | |
| } | |  |  |

* + 1. **Функция get\_Name**

Функция возвращает имя конечного автомата***.***

* + 1. **Функция get\_Id**

Функция возвращает идентификатор конечного автомата***.***

* + 1. **Функция AddState**

Функция добавляет состояние в конечный автомат с заданным именем ***Name*** и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\*.***

* + 1. **Функция get\_States**

Функция возвращает список всех состояний конечного автомата в виде указателя на интерфейс для работы со списком ***IEcoList1\*.*** Элементом списка является указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\*.***

* + 1. **Функция AddEvent**

Функция добавляет событие в конечный автомат с заданными именем ***Name***, уникальным идентификатором ***Id***, флагом пустого события ***isNull*** и возвращает указатель на интерфейс события конечного автомата ***IEcoFSM1Event\*.***

Если ***Id*** равен нулю, то идентификатор события назначается автоматически.

Если ***Id*** не равен нулю и совпадает с существующим ***Id*** событием в списке, то событие не добавляется в список, возвращая нулевой указатель на интерфейс.

* + 1. **Функция get\_Events**

Функция возвращает список всех событий конечного автомата в виде указателя на интерфейс для работы со списком ***IEcoList1\*.*** Элементом списка является указатель на интерфейс события конечного автомата ***IEcoFSM1Event\*.***

* + 1. **Функция AddTransition**

Функция добавляет переход в конечный автомат с заданными событием ***Event***, исходным состоянием ***Source***, конечным состоянием ***Target*** и возвращает указатель на интерфейс перехода конечного автомата ***IEcoFSM1Transition\*.***

* + 1. **Функция get\_Transitions**

Функция возвращает список всех переходов конечного автомата в виде указателя на интерфейс для работы со списком ***IEcoList1\*.*** Элементом списка является указатель на интерфейс перехода конечного автомата ***IEcoFSM1Transition\*.***

* + 1. **Функция AddTransitionById**

Функция добавляет переход в конечный автомат с заданными идентификаторами события ***EventId***, исходного состояния ***SourceId***, конечного состояния ***TargetId*** и возвращает указатель на интерфейс перехода конечного автомата ***IEcoFSM1Transition\*.***

1. **Интерфейс IEcoFSM1State**
   1. **IEcoFSM1State описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(6242C2A9-31D9-4FB9-A1CB-8AC693AD6EB4),  ] | | | |
| interface IEcoFSM1State : IEcoUnknown { | | | |
|  | | | |
| int16\_t | | ***get\_Id*** | ([in] void); |
|  | |  |  |
| char\_t\* | | ***get\_Name*** | ([in] void); |
|  | | |  |
| void | | ***set\_Name*** | ([in] char\_t\* Name); |
|  | | | |
| bool\_t | | ***IsInitial*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Initial*** | ([in] bool\_t isInitial); |
|  | | | |
| bool\_t | | ***IsFinal*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Final*** | ([in] bool\_t isFinal); |
|  | |  |  |
| bool\_t | | ***IsTrap*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Trap*** | ([in] bool\_t isTrap); |
|  | |  |  |
| void\* | | ***get\_Parameter*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Parameter*** | ([in] void\* value); |
|  | | | |
| } | |  |  |

* + 1. **Функция get\_Id**

Функция возвращает идентификатор состояния конечного автомата***.***

* + 1. **Функция get\_Name**

Функция возвращает имя состояния конечного автомата***.***

* + 1. **Функция set\_Name**

Функция устанавливает(назначает) имя состояния конечного автомата***.***

* + 1. **Функция IsInitial**

Функция проверяет состояние конечного автомата и возвращает значение равное единице «1» в случае, если состояние является начальным. В остальных случаях возвращает ноль «0»***.***

* + 1. **Функция set\_Initial**

Функция устанавливает(задает) значение начального состояния конечного автомата в случае, если значение равно единице «1»***.*** В остальных случаях, принимает значение равное ноль «0».

* + 1. **Функция IsFinal**

Функция проверяет состояние конечного автомата и возвращает значение равное единице «1» в случае если состояние является конечным, в остальных случаях возвращает ноль «0»***.***

* + 1. **Функция set\_Final**

Функция устанавливает(задает) значение конечного состояния конечного автомата в случае, если значение равно единице «1»***.*** В остальных случаях, принимает значение равное ноль «0».

* + 1. **Функция IsTrap**

Функция проверяет состояние конечного автомата и возвращает значение равное единице «1» в случае если состояние является ловушкой, в остальных случаях возвращает ноль «0»***.***

* + 1. **Функция set\_Trap**

Функция устанавливает(задает) значение ловушки состояния конечного автомата в случае, если значение равно единице «1»***.*** В остальных случаях, принимает значение равное ноль «0».

* + 1. **Функция get\_Parameter**

Функция возвращает указатель на параметры пользователя***.***

* + 1. **Функция set\_Parameter**

Функция устанавливает(сохраняет) указатель на параметры пользователя.

1. **Интерфейс IEcoFSM1Event**
   1. **IEcoFSM1Event описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(1BFBB309-B33F-4E86-AABE-0C41ED5E256C),  ] | | | |
| interface IEcoFSM1Event : IEcoUnknown { | | | |
|  | | | |
| char\_t\* | | ***get\_Name*** | ([in] void); |
|  | | |  |
| int16\_t | | ***get\_Id*** | ([in] void); |
|  | |  |  |
| bool\_t | | ***IsNull*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Null*** | ([in] bool\_t isNull); |
|  | |  |  |
| void\* | | ***get\_Parameter*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Parameter*** | ([in] void\* value); |
|  | | | |
| } | |  |  |

* + 1. **Функция get\_Name**

Функция возвращает имя события конечного автомата***.***

* + 1. **Функция get\_Id**

Функция возвращает идентификатор события конечного автомата***.***

* + 1. **Функция IsNull**

Функция проверяет событие конечного автомата и возвращает значение равное единице «1» в случае если событие является пустым, в остальных случаях возвращает ноль «0»***.***

* + 1. **Функция set\_Null**

Функция устанавливает(задает) пустое значение события конечного автомата в случае, если значение равно единице «1»***.*** В остальных случаях, принимает значение равное ноль «0».

* + 1. **Функция get\_Parameter**

Функция возвращает указатель на параметры пользователя***.***

* + 1. **Функция set\_Parameter**

Функция устанавливает(сохраняет) указатель на параметры пользователя.

1. **Интерфейс IEcoFSM1Transition**
   1. **IEcoFSM1Transition описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(00C3032A-56F2-4474-B4FD-6F837D907909),  ] | | | |
| interface IEcoFSM1Transition : IEcoUnknown { | | | |
|  | | | |
| char\_t\* | | ***get\_Name*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Name*** | ([in] char\_t\* Name); |
|  | | |  |
| int16\_t | | ***get\_Id*** | ([in] void); |
|  | |  |  |
| void\* | | ***get\_Parameter*** | ([in] void); |
|  | |  |  |
| void | | ***set\_Parameter*** | ([in] void\* value); |
|  | | | |
| IEcoFSM1Event\* | | ***get\_Event*** | ([in] void); |
|  | |  |  |
| IEcoFSM1State\* | | ***get\_Source*** | ([in] void); |
|  | |  |  |
| IEcoFSM1State\* | | ***get\_Target*** | ([in] void); |
|  | |  |  |
| } | |  |  |

* + 1. **Функция get\_Name**

Функция возвращает имя перехода конечного автомата***.***

* + 1. **Функция set\_Name**

Функция устанавливает(назначает) имя перехода конечного автомата***.***

* + 1. **Функция get\_Id**

Функция возвращает идентификатор перехода конечного автомата***.***

* + 1. **Функция get\_Parameter**

Функция возвращает указатель на параметры пользователя***.***

* + 1. **Функция set\_Parameter**

Функция устанавливает(сохраняет) указатель на параметры пользователя.

* + 1. **Функция get\_Event**

Функция возвращает указатель на интерфейс события конечного автомата ***IEcoFSM1Event\*.***

* + 1. **Функция get\_Source**

Функция возвращает указатель на интерфейс исходного состояния конечного автомата ***IEcoFSM1State\*.***

* + 1. **Функция get\_Target**

Функция возвращает указатель на интерфейс конечного состояния конечного автомата ***IEcoFSM1State\*.***

1. **Интерфейс IEcoFSM1EventProcessing**
   1. **IEcoFSM1EventProcessing описание на ECO IDL**

|  |
| --- |
| **ECO IDL** |
| import "IEcoBase1.h" | | | |
|  | | | |
| [  object,  uguid(A768CE5B-8B4C-4873-80F9-8FCC5365A855),  ] | | | |
| interface IEcoFSM1EventProcessing : IEcoUnknown { | | | |
|  | | | |
| IEcoFSM1StateMachine\* | | ***get\_StateMachine*** | ([in] void); |
|  | | |  |
| IEcoFSM1State\* | | ***ResetStateMachine*** | ([in] void); |
|  | | | |
| IEcoFSM1State\* | | ***get\_CurrentState*** | ([in] void); |
|  | | | |
| IEcoFSM1State\* | | ***IsInitialState*** | ([in] void); |
|  | |  |  |
| IEcoFSM1State\* | | ***IsFinalState*** | ([in] void); |
|  | |  |  |
| IEcoFSM1State\* | | ***InvokeByEvent*** | ([in] IEcoFSM1Event\* Event); |
|  | | | |
| IEcoFSM1State\* | | ***InvokeById*** | ([in] int16\_t Id); |
|  | |  |  |
| IEcoFSM1State\* | | ***InvokeByName*** | ([in] char\_t\* Name); |
|  | | | |
| } | |  |  |

* + 1. **Функция get\_StateMachine**

Функция возвращает указатель на интерфейс конечного автомата ***IEcoFSM1StateMachine\*.***

* + 1. **Функция ResetStateMachine**

Функция сбрасывает конечный автомат в начальное состояние и возвращает указатель на интерфейс начального состояния конечного автомата ***IEcoFSM1State\*.***

* + 1. **Функция get\_CurrentState**

Функция возвращает указатель на интерфейс текущего состояния конечного автомата ***IEcoFSM1State\*.***

* + 1. **Функция isInitialState**

Функция проверяет текущее состояние конечного автомата и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\**** в случае если состояние является начальным, в остальных случаях возвращает нуль «null»***.***

* + 1. **Функция isFinalState**

Функция проверяет текущее состояние конечного автомата и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\**** в случае если состояние является конечным, в остальных случаях возвращает ноль «null»***.***

* + 1. **Функция InvokeByEvent**

Функция обрабатывает событие по заданному указателю на интерфейс события конечного автомата ***Event*** и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\**** после обработки***.***

* + 1. **Функция InvokeById**

Функция обрабатывает событие по заданному идентификатору события конечного автомата ***Id*** и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\**** после обработки***.***

* + 1. **Функция InvokeByName**

Функция обрабатывает событие по заданному имени события конечного автомата ***Name*** и возвращает указатель на интерфейс состояния конечного автомата ***IEcoFSM1State\**** после обработки***.***

1. **В разработке …**

# **Приложение А**