Санкт-Петербургский государственный университет информационных технологий, механики и оптики
Кафедра “Компьютерные технологии”

А.С. Наумов

Объектно-ориентированное программирование с явным выделением состояний

Бакалаврская работа

Научный руководитель: докт. техн. наук, профессор Шалыто А.А.

Санкт-Петербург
2004

АННОТАЦИЯ

В данной работе вначале был выполнен анализ объектно-ориентированных методов, применяемых в рамках SWITCH-технологии, и сформулированы проблемы, которые необходимо устранить для повышения эффективности указанной технологии. На основе этого анализа предложен подход, заключающийся в динамическом создании и модификации автоматов, который позволил решить отмеченные выше проблемы. Также введено различие между синхронными и асинхронными вариантами взаимодействия с автоматами.

Спроектирована и реализована библиотека “Логическое управление” (LoCoL – LOgical COntrol Library), позволяющая достаточно эффективно применять “автоматное программирование” для реализации поведения объектов со сложным поведением в проектах, создаваемых с использованием объектно-ориентированной парадигмы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В работе [1] предложен подход к созданию программ для систем логического управления на основе использования конечных автоматов. Этот подход был назван “Switch-технология” (по названию оператора выбора в языках программирования с С-подобным синтаксисом) или “автоматное программирование”. Термин “объектно-ориентированное программирование с явным выделением состояний” был введен в работе [17].

Отметим, что основные идеи вышеназванного подхода не новы. Однако данная тема является актуальной, о чем свидетельствует, например, использование диаграмм состояний в языке UML для описания поведения объектов [1] или разработка языка *AsmL* корпорации *Microsoft* [23].

Данное направление достаточно активно исследуется в СПбГУ ИТМО [23]. При этом на текущий момент создано около 50 проектов, использующих “автоматное программирование”, в которых авторы предлагают различные решения.

Целью настоящей работы является проведение анализа вышеупомянутых проектов, выделение проблем и предложение путей их преодоления.

По мнению автора, решением этих проблем является разработка и использование библиотеки “Логическое управление” *LoCoL* (LOgical COntrol Library), основанной на новом взгляде на “автоматное программирование”, заключающемуся в динамическом создании и модификации автоматов.

1. АНАЛИЗ СУЩЕСТВУЮЩИХ ПРОЕКТОВ. ПРОБЛЕМЫ И ИХ РЕШЕНИЯ

Выполним анализ проектов (http://is.ifmo.ru, раздел “Проекты”), использующих “автоматный подход”, и выделим характерные для них проблемы, снижающие эффективность применения объектно-ориентированного программирования. В результате анализа предложены варианты решения указанных проблем безотносительно конкретной реализации.

1.1. Автоматы и объектно-ориентированное программирование

Начнем с концептуальных вопросов, связанных с проектированием. Один из самых спорных вопросов, характерных практически для всех работ на основе SWITCH-технологии, это ее связь с объектно-ориентированным программированием. Другими словами, необходимо выяснить, чем является автомат, как абстракция централизованной логики работы программы или ее части? Существует несколько ответов на этот вопрос. Проклассифицируем их.

1. Автоматы, как методы класса, – процедурный стиль программирования [18].
2. Автоматы, как классы без использования наследования. При этом автоматы выделяются как сущность (класс), в которой не используются преимущества объектно-ориентированного программирования [6].
3. Автоматы, как классы с использованием наследования. Этот подход более удобен, так как часть функциональности, общей для всех автоматов может быть вынесена в базовый класс.

    3.1. В работе [19] предложена библиотека *STOOL* (Switch-Technology Object Oriented Library), в которой не только автомат,

но и его составляющие имеют базовые классы, например, состояние, входное воздействие, выходное воздействие.

3.2. В работе [12] предложена еще одна библиотека для объектно-ориентированной реализации автоматов, названная *Auto-Lib*. В ней также имеются базовые классы для составных частей автомата, например, состояния и события. Заметим, что в этой библиотеке введены такие сущности, как, например, `DescriptedState` (состояние с описанием). Такой подход обеспечивает автоматизацию ведения протокола работы автоматов.

3.3. В работе [12] предложен подход, позволяющий “собирать” простые автоматы из наследников базовых классов “Состояние автомата” и “Переход между состояниями”. Эта работа показывает, как можно обеспечить изоморфизм между текстом программы и графом переходов при наличии в нем групповых переходов.

4. Кроме того, существуют примеры использования паттернов проектирования [1, 6].

Отметим, что ни одна из библиотек не предоставляет возможности полного конфигурирования автомата – несмотря на то, что основные части автомата проектируются как отдельные сущности, которые некоторым образом регистрируются в автомате, они в совокупности еще не самодостаточны как данные, полностью определяющие логику работы автомата.

Итак, какой же подход предлагается в данной работе? Бесспорно, автомат можно рассматривать как класс, но в отличие от авторов, проанализированных выше проектов, эта сущность является лишь механизмом, который оперирует с данными, определяющими логику работы автомата. В качестве этих данных могут выступать экземпляры классов “Состояние”, “Переход” и т.д.

Сформулируем преимущества такого подхода.

1.2. Отсутствие шаблонов и стандартов

Автоматное программирование, обсуждаемое в этой работе, находится в стадии подъема и развития. Появляется всё больше работ по этой тематике, и в большинстве из них авторы хотят предложить что-то новое, которое, по их мнению, должно улучшить реализацию этой концепции (зачастую именно так и происходит).

Я считаю, что в ближайшее время вряд ли появятся стандарты и шаблоны, так как технология будет еще определенное время развиваться. Что же касается данной работы, то она ни в коем случае не претендует на “истину в последней инстанции”. Я надеюсь лишь, что какие-то моменты и идеи, упомянутые здесь, помогут другим авторам в будущем совершенствовать “SWITCH-технологию”, как помогли мне уже существующие работы.

1.3. Изоморфность кода. Хорошо или плохо?

В этом разделе речь пойдет о таком плюсе SWITCH-технологии, как изоморфность кода программы графу переходов, строящемуся на этапе проектирования.

Напомню, что основной метод автомата на каждом шаге выполняет следующие действия:

```
текущее_состояние_резерв = текущее_состояние;
switch ( текущее_состояние )
{
          case номер_состояния_0:
          {
                    if ( условие_перехода_0 )
                    {
                              действия_на_переходе_0();
                              текущее_состояние = новый_номер_состояния_0;
                    }
                    else if ( условие_перехода_1 )
                    {
                              действия_на_переходе_0();
                              текущее_состояние = новый_номер_состояния_0;
                    }
                    ...
                    break;
```

```
		}
		...
		case номер_состояния_N:
		{
			if ( условие_перехода_K )
			{
				действия_на_переходе_K();
				текущее_состояние = новый_номер_состояния_K;
			}
			else if ( условие_перехода_L )
			{
				действия_на_переходе_L();
				текущее_состояние = новый_номер_состояния_L;
			}
			...
			break;
		}
}
if (текущее_состояние_резерв != текущее_состояние )
{
		case номер_состояния_0:
		{
			действия_в_состоянии_0();
			break;
		}
		...
		case номер_состояния_N:
		{
			действия_в_состоянии_N();
			break;
		}
}
```

Как следует из приведенного текста, все автоматные методы получаются однотипными. Однако при этом происходит неоправданное разрастание кода. Если автомат рассматривать, как класс, то появляется возможность определить механизм выбора переходов и выполнения действий для всех автоматов лишь однажды, построив библиотечный класс соответствующим образом. Тогда в коде разработчика будет необходимо создать объект этого класса и инициализировать его исходными данными о конкретном автомате. Таким образом, изоморфность будет существовать уже не на уровне кода, действий, а на уровне данных, таких как переход, состояние и т.д.

Итак, в предложенном подходе возможно сохранение изоморфности графа переходов данными, с которыми оперирует класс-автомат, при устранении разрастания кода.

1.4. Динамическое изменение автоматов

Следует отметить, что предлагаемый в этой работе подход, позволяет динамически (в ходе работы программы) изменять автоматы. Здесь под словом “автомат” подразумевается логика работы программы или её части, а не класс, реализующий механизм, который позволяет использовать эту логику в программном коде. В дальнейшем из контекста должно быть понятно, что подразумевается под термином автомат.

Хочу отметить, что пока идёт лишь теоретическое, концептуальное обсуждение проблем, идей, решений. При попытке проекции их на одну из существующих технологий программирования, будет неизбежен возврат к этим вопросам и решение их уже в конкретном контексте.

1.5. Взаимодействие автоматов

Рассмотрим ещё один важный вопрос, который уже не раз поднимался, например, в работах [1, 7, 8]: взаимодействие параллельно работающих автоматов. При этом можно выделить две проблемы:

- взаимодействие автоматов;
- синхронизация автоматов.

Под взаимодействием подразумевается механизм обмена событиями или сообщениями. Введем более общее понятие – уведомление. Его целесообразность будет обоснована немного позже.

Под синхронизацией будем понимать то, что автоматы работают “псевдопараллельно”. Опишу термины, используемые в дальнейшем.

Система автоматов – совокупность логически связанных автоматов. Взаимодействие автоматов возможно лишь в том случае, если они находятся в одной системе автоматов. Гарантия синхронности распространяется только на совокупность автоматов внутри одной системы автоматов.

Менеджер автоматов – механизм, который контролирует (в частности, синхронизирует) систему автоматов. Все взаимодействия между автоматами возможны только с помощью менеджера автоматов. Стоит отметить, что идея менеджера автоматов не нова, и уже была использована, например, в работе [15].

Уведомление – единица взаимодействия автоматов.

Событие – синхронное уведомление. Синхронность понимается в том смысле, что если автомату приходит событие, то он обязан немедленно обработать его. За этим следит менеджер автоматов. По сути, этот механизм соответствует вызываемым (или вложенным) автоматам, которые, в отличие от работы [18], будут рассматриваться как единый вариант взаимодействия автоматов. Кроме того, в отличие от этой работы, вводится взаимодействие при помощи сообщений [1, 7, 8].

Сообщение – асинхронное уведомление. Все сообщения помещаются в очередь сообщений менеджера системы автоматов. При каждом такте извлекается верхнее сообщение для каждого автомата и передаётся ему на обработку. Заметим, что автомат обрабатывает за один такт либо одно событие, либо одно сообщение.

В различных работах, приведенных на сайте http://is.ifmo.ru/ в разделе “Проекты”, почему-то оставляли только один из указанных видов взаимодействия, игнорируя другой. Я считаю, что разработчику необходимо дать больше возможностей и предоставить более гибкую систему взаимодействия автоматов. Отмечу также, что в отличие от работ [1, 7, 8], очередь сообщений не очищается в конце каждого такта. Это тоже сделано из соображений повышения гибкости системы, ведь приводимый в этих работах довод, что генерация событий может происходить быстрее их обработки, скорее всего, свидетельствует о неправильно разработанной логике поведения системы взаимосвязанных автоматов.

Теперь перейдем к рассмотрению проблемы синхронизации.

В работе [7] предложен вариант разбиения автоматной процедуры (вид которой упомянут в разд. 1.3) на две части: процедуру выбора перехода (A-процедура) и процедуру, обеспечивающую переход и действия на нём (S-

процедура). Для всех автоматов в системе сначала выполняются A-процедуры, а затем S-процедуры. Но в этой работе нет синхронных уведомлений (событий), а в данной работе они есть. Поэтому в случае, когда выходное воздействие автомата A_i – посылка события автомату A_j, возникает нежелательный эффект (в системе будут как минимум два автомата, у которых выполняются разные процедуры A и S).

В работе [1] предложен другой механизм взаимодействия автоматов: сначала для всех автоматов выполняется переход-действие (TA – Transition-Action), а затем для них выполняется обновление переменной состояния (U – Update). Такой подход не вызывает описанной выше проблемы. Однако стоит отметить, что в данной ситуации существует неразрешимая в общем виде проблема.

Проиллюстрируем её на простом примере. Пусть у автоматов A_0 и A_1 в очереди есть сообщения m_{10} и m_{11} соответственно. Предположим, что на следующем такте менеджер автоматов вызываем сначала автомат A_0 с сообщением m_{10}. Тогда автомат успешно отрабатывает стадию TA, и после этого запускается автомат A_1 с сообщением m_{11}. Автомат A_1 в одном из выходных воздействий посылает событие автомату A_0, который обработать его на этом такте уже не сможет.

Попытаемся проанализировать проблему с точки зрения проектирования. Посылка события, по сути, вызов вложенного автомата. Договоримся, что вложенный автомат имеет больший номер, чем автомат, в который он вложен. Так как, в принципе, не должно быть циклов в ориентированном графе вложенности автоматов, получается частично упорядоченное множество автоматов по критерию вложенности. Если вызывать автоматы с сообщениями в порядке возрастания их номеров, то возникновение ошибки, описанной в предыдущем абзаце, будет свидетельствовать об ошибке проектирования.

1.6. Связь с внешней средой

Очень важным вопросом, которым, к сожалению, задавались далеко не все разработчики, применявшие автоматный подход, является связь системы автоматов с внешней средой. Зачастую программный код логического управления тесно сплетен с самими объектами управления. В таком случае очень трудно провести границу между системой управления и подчиненным ей объектом управления. В данной работе уже упоминалось, что это снижает положительные качества используемого подхода. В идеальном случае библиотека должна предоставить только настраиваемую систему управления, способную работать с объектами управления, на которые накладываются минимальные ограничения и требования. При этом взаимодействие должно быть централизовано (например, посредством менеджера автоматов) и строго декларировано (например, функциональностью того же самого менеджера).

Основные аспекты взаимодействия с внешней средой упоминаются в большинстве параграфов данной главы. Здесь перечислены лишь основные вопросы, к конкретным решениям которых вернёмся в дальнейшем:

- возможность динамически изменять систему автоматов;
- возможность пользовательской проверки автоматов в системе;
- посылка через менеджер уведомления конкретному автомату в системе;
- проверка состояния конкретного автомата в системе;
- гибкая возможность настройки протоколирования системы;
- гибкая возможность установки входных переменных и выходных воздействий;
- хранение описания системы автоматов;
- возможность установки таймеров на посылку уведомлений.

1.7. Протоколирование

К сожалению, ошибки, допускаемые на различных этапах проектирования и программирования, являются неизбежными. Во всем мире прилагаются большие усилия для упрощения их обнаружения и исправления. Сложные системы отладки и тестирования, интегрированные в среды разработки, и многие другие решения призваны облегчить путь от технического задания до финального релиза программного продукта. Автоматное программирование вносит свой вклад в решение этой проблемы. Он основан на том факте, что в автомате сконцентрирована “чистая” логика работы некоторой сущности. В этом аспекте легко реализовать протоколирование всех элементарных операций, происходящих в системе, например, переход между состояниями или опрос входной переменной. Такого рода информация относительно легко позволяет обнаружить ошибки, как разработки, когда логика спроектирована неправильно, так и программирования, когда система автоматов неизоморфна графам переходов.

Большинство проанализированных проектов были созданы таким образом, что код вызова методов протоколирования разработчик вынужден писать сам, причём в строго определённом месте программы. Очевидно, что полноценной выгоды при этом не получается, так как, во-первых, необходимо проделать лишнюю работу, а во-вторых, что более важно, можно при этом допустить ошибку. Поэтому необходимо сделать протоколирование готовым инструментом библиотеки, не отнимая у пользователя возможности гибкой настройки вида протоколирования.

Предлагается такой подход к реализации протоколирования. Механизм протоколирования должен реализовывать методы реакции на происходящие элементарные операции в системе автоматов. Определим набор этих операций:

- начало элементарного цикла;
- запуск автомата с некоторым уведомлением;
- опрос входной переменной;

- сравнение переданного уведомления;
- сравнение состояния некоторого автомата в системе;
- выполнение перехода;
- вызов выходного воздействия;
- посылка сообщения;
- завершение обработки уведомления автоматом;
- окончание элементарного цикла.

1.8. Входные переменные и выходные воздействия

Одними из центральных понятий “автоматного программирования” являются входные переменные, обозначаемые обычно как xNN и выходные воздействия (zNN), где NN – некоторое число. Входные переменные могут при необходимости “опрашиваться”. При этом они возвращают значения булевого типа (true/false). Это необходимо при проверке выполнения условий перехода. Заметим, что значения входных переменных не зависят от порядка и количества обращений к ним – они не изменяют состояние внешней среды. Выходные воздействия – это методы, которые вызываются при выполнении некоторых действий (например, при переходе из одного состояния в другое). Они изменяют состояние внешней среды. По сути, на вход автомат получает входные переменные, значения состояний других автоматов системы и события, а на выходе формируются различные выходные воздействия.

Типичная схема взаимодействия автомата представлена на рис. 1. Источниками событий (в предлагаемой библиотеке – уведомлений) могут быть как сами автоматы, так и внешний код разработчика. Заметим, что посылка уведомлений, равно, как опрос состояний других автоматов, осуществляется только через менеджера автоматов.

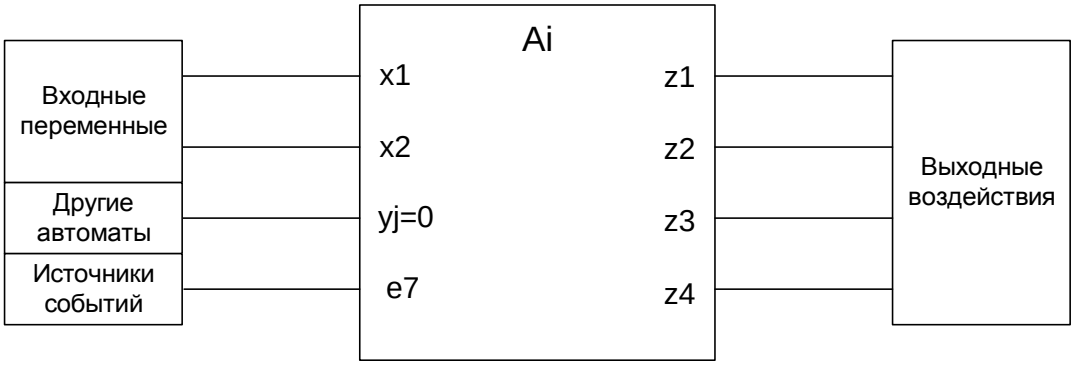

Рис. 1. Типичная схема взаимодействия автомата

Итак, автомат непосредственно работает только с входными переменными и выходными воздействиями. Поэтому актуальным является вопрос, насколько тесно должен быть связан с ними автомат. Вспомним, что в большинстве указанных выше работ входные переменные и выходные воздействия жестко привязаны к самому автомату. Это, естественно, снижает возможность повторного использования кода. Если же следовать обозначенным выше принципам проектируемой библиотеки, таким как динамическое изменение автоматов и “автомат – это только логика”, то необходимо предоставить пользователям возможность гибкого управления входными переменными и выходными воздействиями конкретного автомата. Забегая вперёд, отмечу, что *.NET* позволяет очень изящно решить эту проблему.

1.9. Хранение описания системы автоматов

Еще одним плюсом разрабатываемой библиотеки можно назвать возможность сохранения и восстановления описания системы автоматов. Действительно, так как вся логика функционирования системы представлена набором данных (а если точнее – связанным набором объектов), а не набором инструкций (команд), оперировать изменением которых значительно сложнее (а порой невозможно), то очевидным решением было бы предоставить пользователю библиотеки возможность сохранения и восстановления этих данных. При этом

желательно иметь возможность обмена информацией о системе автоматов не только между разными запусками одной программы, но и между различными приложениями. Одним из очевидных применений этого механизма может быть генерация логики одним приложением, принимающим на вход, например, диаграмму классов, созданную в *Microsoft Visio*, и использование сохраненного образа этой логики другим.

Стоит задаться вопросом, следует ли позволять сохранение состояния системы, а не только ее описание. Я склоняюсь к мнению, что это небезопасно и в общем случае такой инструмент принесет больше вреда, чем пользы. Действительно, практически невозможно создать программу, состояние которой полностью описывалось бы системой автоматов. Я думаю, что это вполне очевидно любому человеку, написавшему когда-нибудь программный продукт, содержащий более “шестнадцати экранов” кода. Исходя из этого, восстановление состояния системы автоматов может привести к расхождениям между ней и внешней по отношению к ней средой, что, естественно, нарушит нормальную работу приложения.

2. БИБЛИОТЕКА *LOCOL*. ЭТАП ПРОЕКТИРОВАНИЯ

В этой главе будет сделана попытка спроектировать библиотеку “логическое управление”, реализующую идеи, которые были оглашены ранее. Будут выделены интерфейсы и сущности, которые в конкретной реализации, скорее всего, представимы в виде классов. Будет описана функциональность этих сущностей (данные, которые они инкапсулируют и методы обработки этих данных) и возможности контрактов (механизмы, предоставляемые ими).

2.1. Интерфейсы

2.1.1. Изменяемый

В библиотеке желательно знать, когда будут изменяться некоторые сущности (например, набор переходов автомата), для того, чтобы адекватно отреагировать на эти действия. Напомним, что это требование следует из возможности динамического изменения системы автоматов.

2.1.2. Проверяемый

Желательно, чтобы сущности представляющие собой логические части системы автоматов могли сами себя проверять (тестировать), чтобы сообщать пользователю об ошибках, допущенных при построении той или иной системы.

2.1.3. Проверяющий автоматы

Желательно предоставить механизм, с помощью которого пользователь библиотеки смог бы добавить свои варианты проверки (тестирования) автоматов на корректность. Например, определить потенциальную достижимость состояний автомата.

2.1.4. Протоколирующий

Сущность, реализующая этот контракт, имеет возможность протоколировать работу системы автоматов.

2.1.5. Контролирующий детальность протоколирования

Желательно иметь возможность гибкого управления детальностью протоколирования. Данный контракт позволяет реализовать этот механизм.

2.1.6. Сериализуемый

Кроме того, в библиотеке желательно определение механизма сохранения и загрузки описания системы автоматов. Предполагается, что для этого сущности должны реализовать данный контракт.

2.2. Основные сущности библиотеки

2.2.1. Именованный объект

Большинство объектов библиотеки именованы. Другими словами именованный объект – один из базовых классов библиотеки, предок большого количества классов. Представляет собой пару:

- идентификационный номер объекта;
- имя объекта.

Заметим, что именованный объект реализует контракт “Сериализуемый”.

2.2.2. Система автоматов и менеджер автоматов

Как было отмечено выше, система автоматов – это совокупность логически связанных автоматов. Механизмом контроля работы системы и взаимодействия ее с внешней средой является менеджер автоматов. При реализации библиотеки не стоит выделять систему автоматов в отдельную сущность, в отличие от

менеджера автоматов. Последний будет содержать первую в качестве одной из составных частей.

Взаимодействие между автоматами внутри системы и системы с внешней средой осуществляется при помощи механизма уведомлений, который был рассмотрен в первой главе. Таким образом, набор уведомлений является общим для системы и действителен только для этой системы. Очевидным было бы потребовать декларации набора возможных уведомлений. Заметим, что именно уведомлений, а не событий и сообщений, так как будем считать, что уведомление несет информативную часть, а его вид (событие или сообщение) – это всего лишь вариант “доставки” до конечного автомата. Уведомление $n0$ зарезервировано системой.

Кроме контроля взаимодействия, менеджер следит за рентабельностью системы, не позволяя за один элементарный цикл запустить один и тот же автомат больше одного раза, а также предоставляя во время цикла номера состояний автоматов на момент начала цикла. Заметим, что запуск автомата не внутри цикла автоматически начинает новый цикл, который также автоматически завершается сразу же после того, как автомат отработает.

Менеджер предоставляет возможность отправки уведомления наборам автоматов через заданный промежуток времени. Количество такого рода таймеров практически не ограничено.

И, наконец, менеджер ведет автоматическое протоколирование и способен проверять автоматы на адекватность внешними сущностями, реализующими соответствующий контракт.

Данные:

- набор автоматов, представляющий собой систему автоматов, которую контролирует менеджер;
- набор зарегистрированных в системе уведомлений.

Методы:

- начало элементарного цикла;

- окончание элементарного цикла;
- посылка события автомату;
- посылка сообщения автомату;
- запуск автомата (при этом автомату передается первое сообщение из очереди, либо в случае, если очередь сообщений пуста, передается сообщение *m0*);
- установка таймера на посылку события набору автоматов через заданный промежуток времени;
- установка таймера на отправку сообщения набору автоматов через заданный промежуток времени;
- исключение ранее установленного таймера;
- регистрация механизма протоколирования;
- исключение ранее зарегистрированного механизма протоколирования;
- регистрация механизма, проверяющего автоматы;
- исключение ранее зарегистрированного механизма, проверяющего автоматы.

Контракты: изменяемый; проверяемый; сериализуемый.

События (в терминах обычных технологий программирования), которыми оповещаются механизмы протоколирования, реализация зависит от выбора технологии:

- начало элементарного цикла;
- запуск автомата с некоторым уведомлением;
- опрос входной переменной;
- сравнение переданного уведомления;
- сравнение состояния некоторого автомата в системе;
- выполнение перехода;
- вызов выходного воздействия;
- посылка сообщения;

- завершение обработки уведомления автоматом;
- окончание элементарного цикла.

2.2.3. Автомат

Компонента управления.

Данные:

- набор состояний;
- текущее состояние (очевидно, входит в набор состояний автомата);
- набор групп состояний;
- набор переходов;
- набор зарегистрированных входных переменных;
- набор зарегистрированных выходных воздействий.

Методы:

- регистрация класса, содержащего входные переменные для данного автомата;
- регистрация класса, содержащего выходные воздействия для данного автомата;
- удаление из списка зарегистрированных классов, содержащих входные переменные для данного автомата;
- удаление из списка зарегистрированных классов, содержащих выходные воздействия для данного автомата;
- регистрация сущности, проверяющей данный автомат;
- регистрация механизма, проверяющего автоматы;
- исключение ранее зарегистрированного механизма, проверяющего автоматы.

Контракты: изменяемый; проверяемый; сериализуемый.

2.2.4. Состояние

Класс, описывающий одно из возможных состояний автомата. Является именованным объектом с набором действий, которые выполняются при переходе автомата в данное состояние.

Контракты: изменяемый, проверяемый, сериализуемый.

2.2.5. Группа состояний

Класс, наследуемый от именованного объекта. Содержит набор состояний, которые входят в данную группу.

Контракты: изменяемый, сериализуемый.

2.2.6. Переход

Класс, наследуемый от именованного объекта, описывающий один из возможных переходов автомата.

Данные:

- идентификационный номер состояния (или группы состояний), из которого (которой) ведет данный переход;
- флаг, указывающий из группы состояний или конкретного состояния, ведет данный переход;
- идентификационный номер состояния, в который ведет данный переход;
- приоритет перехода;
- условие, выполнение которого необходимо, для осуществления перехода;
- действия, выполняемые на переходе.

Контракты: изменяемый; проверяемый; сериализуемый.

2.2.7. Входная переменная

Именованный объект, представляющий собой информацию о входной переменной автомата.

Контракт: сериализуема.

2.2.8. Выходное воздействие

Именованный объект представляющий собой информацию о выходном воздействии автомата.

Контракт: сериализуем.

2.2.9. Уведомление

Именованный объект. Содержит информацию о типе уведомления (событие или сообщение).

Контракт: сериализуемо.

2.2.10. Действие автомата

В библиотеке возможны два вида действий: выходное воздействие и посылка уведомления. Необходима сущность или контракт для описания действия автомата.

2.3. Дополнительные сущности библиотеки

2.3.1. Коллекции

Отметим, коллекции, каких сущностей необходимы в библиотеке. Необходимость написания собственного кода для них зависит от конкретной реализации:

- упорядоченный набор действий автоматов;
- набор автоматов;

- набор входных переменных;
- набор уведомлений;
- набор выходных воздействий;
- набор групп состояний;
- набор состояний;
- набор переходов.

2.3.2. Выражения

Выполнение перехода в общем случае зависит от истинности некоторого логического выражения. Для динамического построения таких выражений необходим соответствующий набор сущностей. Определим функциональность этого набора:

- логическое выражение “и”;
- логическое выражение “или”;
- логическое выражение “исключающее или”;
- логическое выражение “не”;
- проверка входной переменной;
- проверка уведомления, с которым запущен автомат;
- проверка состояния некоторого автомата в системе.

2.3.3. Протоколирование

Желательно реализовать несложные механизмы протоколирования и управления его детальностью.

2.3.4. Проверка автомата

Желательно реализовать один или несколько механизмов проверки автоматов (контракт “Проверяющий автомат”).

3. БИБЛИОТЕКА LOCOL. РЕАЛИЗАЦИЯ

3.1. Выбор технологии реализации: .NET

Летом на конференции *Professional Developer's Conference 2000* компания *Microsoft* представила миру платформу *.NET*, которая предлагает новый путь разработки программного обеспечения [21, 23, 24]. В основе технологии лежит *Common Language Runtime (CLR)*, которая предоставляет исполняемому коду определенный набор сервисов. Рассмотрим основные понятия *.NET* и сервисы *CLR*, благодаря которым в настоящей работе была выбрана именно эта технология.

3.1.1. Автоматическое управление ресурсами

Одна из наиболее частых ошибок заключается в том, что приложение не освобождает ресурсы или пытается их использовать после освобождения. Среда *CLR* автоматически отслеживает использование ресурсов, гарантируя их своевременное освобождение.

3.1.2. Выполнение на многих платформах

Сегодня имеются много различных версий *Windows*. Написанное и собранное .NET-приложение сможет выполняться на любой платформе (не обязательно *Windows*), которая поддерживает *.NET CLR*.

3.1.3. Контроль типов

.NET CLR контролирует доступ к объектам и гарантирует, что доступ производится в соответствии с типом этих объектов. Также среда *CLR* не позволит создать указатель на произвольный адрес в памяти и выполнить код по

этому адресу. Это, в частности, исключает возникновение множества типичных ошибок программирования и классических атак по методу переполнения буфера.

3.1.4. Интеграция языков

Технология *COM* позволяет взаимодействовать компонентам на различных языках программирования. Технология *.NET* интегрирует языки друг с другом. Например, можно создать *C++* класс, который наследует класс, реализованный на языке *Visual Basic*. Это означает, что программисты, использующие библиотеку классов, при выборе языка больше не ограничены языком этой библиотеки. Исключение может быть вызвано из кода, написанного на одном языке, и обработано в коде, написанном на другом языке. Такая интеграция становится возможной, поскольку технология *.NET* определяет и предоставляет систему типов, общую для всех языков.

3.1.5. Общая система типов

Формальная спецификация системы типов, реализованной средой *CLR*, называется *Common Type System* (*CTS*). Основная задача системы типов – обеспечение интеграции языков. Одно из правил *CTS* гласит, что все базовые классы должны наследовать класс *System.Object*.

3.1.6. Сборки

Технология *.NET* использует новую модель расположения приложений, которая упрощает инсталляцию и отслеживание версий. Ключевое понятие этой модели – сборка (*assembly*). Сборка – это набор ресурсов и типов, а также метаданные, описывающие типы и методы. Таким образом, сборка – это самоописанный компонент. Основное преимущество таких компонентов в том, что для их использования не нужны никакие другие файлы.

3.1.7. Система безопасности

Среда *CLR* предоставляет сервисы безопасности, контролирующие доступ пользователей к ресурсам и действия программ. Поскольку *CLR* используется, чтобы загружать код, создавать объекты и вызывать методы, этот код может управлять безопасностью и предписывать политики безопасности.

3.1.8. Промежуточный язык и JIT-компилятор

После компиляции исходных текстов создаются *EXE*- или *DLL*-файлы, которые отличаются от обычных *PE*-файлов. Код в получаемых файлах не является командами процессора x86 или другим машинным кодом, компилятор создает код на Промежуточном Языке *Microsoft* (*Microsoft Intermediate Language - MSIL*). Кроме того, такой файл содержит метаданные, которые используются средой *CLR* для обнаружения и загрузки типов из файла, расположения объектов в памяти, вызова методов, управления ссылками, перевода *MSIL* в машинные коды, контроля за безопасностью и множества других задач.

MSIL – это процессоронезависимый промежуточный язык более высокого уровня, чем большинство машинных языков. Он понимает типы объектов и имеет инструкции для создания и инициализации объектов, вызова виртуальных методы и непосредственной манипуляции элементами массива. Этот язык даже имеет инструкции, которые оперируют исключениями. Как и любой другой машинный язык, *MSIL* может быть написан на ассемблере. *Microsoft* предоставляет ассемблер и дизассемблер для *MSIL*.

Перед тем, как выполнять управляемый код среда *CLR* должна сначала скомпилировать управляемые *MSIL* инструкции в инструкции процессора. Здесь возникает типичная проблема: когда пользователь запускает программу, он не намерен ждать, пока вся программа скомпилируется, тем более что большинство функций программы не будут вызваны. Поэтому среда *CLR* компилирует *MSIL* код в инструкции процессора, когда функции непосредственно вызваны. Всякий

раз, когда такая функция будет вызываться в будущем, сразу выполняется машинный код (при этом компилятор не вовлекается в процесс). Поскольку *MSIL* компилируется только в нужный момент (just-in-time – *JIT*), этот компонент среды *CLR* часто упоминается как *JIT*-компилятор (*JIT-compiler*) или *JITter*.

3.1.9. Эффективность

Программисты на низкоуровневых языках типа *C* или *C++* возможно думают об эффективности выполнения описанной схемы. Ведь обычный код компилируется сразу в инструкции процессора и при вызове просто выполняется. Однако в окружении *CLR*, чтобы выполнить код, *MSIL* должен компилироваться в инструкции процессора в реальном времени, потребляя большое количество памяти и мощности процессора.

Действительно, *.NET*-код выполняется медленнее и имеет больше накладных расходов, чем обычный код. Однако корпорация *Microsoft* предлагает инструмент, позволяющий выполнять полную компиляцию сборки в машинные коды и сохранять результат на диске. Когда сборка загрузится в следующий раз, будет использована эта сохраненная версия и приложение стартует быстрее.

Отметим особо, что в приложении можно совмещать фрагменты так называемых “управляемого” (код, контролируемый средой *CLR*) и “неуправляемого” (код, который не контролируется этой средой) кодов. Неуправляемый код следует использовать для критичных к времени выполнения функций.

3.1.10. Библиотека базовых классов

Над *CLR* в архитектуре *.NET* находится инфраструктура сервисов (*Services Framework*). Эта инфраструктура предоставляет классы, которые могут быть использованы из любого языка программирования. Каждый класс дает доступ к некоторому элементу основной платформы.

.NET – полностью объектно-ориентированная платформа. Программисты могут создавать собственные пространства имен, содержащие их собственные классы. Это значительно упрощает разработку программного обеспечения по сравнению с классическими *Windows* парадигмами программирования.

3.2. Выбор языка реализации: C#

3.2.1. C# сегодня

C# – новый язык программирования, появившийся на свет одновременно с технологией *.NET*. Этот язык вобрал в себя все лучшее из таких популярных языков как *C++*, *Visual Basic*, *Java* и *Object Pascal*. Язык C# обеспечивает быструю разработку, позволяя при этом писать эффективный код [21, 24]. Перечислим основные особенности нового языка:

- автоматическая сборка мусора;
- возможность манипулировать указателями и иметь непосредственный доступ к памяти;
- поддержка свойств и событий;
- поддержка атрибутов;
- встроенная поддержка основных типов (строка, массив и т.д.);
- множественное наследование возможно только от интерфейсов (как в технологии *Java*);
- поддержка *C API*, *Windows API* и *COM+* на уровне языка;
- поддержка сборок;
- контроль типов;
- автоматическая инициализация переменных.

Атрибуты являются новым механизмом. С их помощью можно ввести новые типы описательной информации для классов и методов и получать эту информацию во время выполнения. Единственное чего не хватает в новом языке – это шаблонов, так полюбившихся программистам на *C++* за последние несколько

лет. Стоит упомянуть о том, что корпорация *Microsoft* в 2003 г. объявила о выпуске следующей версии языка программирования C#. Эти нововведения появятся в версии *Visual Studio for Yukon.*

3.2.2. Будущие возможности C#

Язык, который за короткий срок стал очень популярен среди программистов во всём мире, имеет явные шансы на дальнейшее развитие и ещё больший успех. Перечислим ожидаемые нововведения, о которых корпорация *Microsoft* уже объявила:

- шаблоны. В языке C# будет включена высокопроизводительная версия шаблонов с безопасными типами, мало отличающаяся в синтаксисе и сильно отличающаяся в реализации от шаблонов, применяемых в языке *C++*, и шаблонов, предлагаемых для языка программирования *Java.*
- итераторы. Сегодня, если классы хотят поддерживать итерирование, используя конструкцию цикла `foreach`, они должны реализовать интерфейс `IEnumerator`. Итераторы выполняют рутинную работу по реализации `IEnumerator` от имени вашей программы, обеспечивая существенное увеличение продуктивности разработчика.
- анонимные методы. Анонимные методы – это другая практическая языковая конструкция, которая дает возможность программистам создавать блоки кода, которые могут быть инкапсулированы в делегаты и выполняться позже.
- неполные типы. В то время как хорошей практикой объектно-ориентированного программирования является сохранение всего исходного кода типа в одном файле, иногда ограничения производительности вынуждают отступать от этого правила. Неполные типы дают вам возможность разбить исходные типы, состоящие из

большого количества исходного кода, на несколько файлов для облегчения разработки и обслуживания.

3.3. Коллекции

Опыт показывает, что наличие строго типизированных коллекций снижает количество ошибок, улучшает читабельность кода и упрощает процесс разработки. К сожалению, на данный момент в среде *CLR* отсутствуют шаблоны, которые идеально подходят для выполнения подобной функции (выше было отмечено, что новый стандарт, уже утвержденный *ECMA* (*European Computer Manufacturer's Association*) и *ISO* (*International Standards Organization*), будет содержать шаблоны). Следовательно, приходится выбирать между нетипизированными коллекциями (работающими с объектами типа *System.Object*) и избыточным повтором кода. В библиотеке сделан выбор в пользу второго. Строгая типизация практически однотипных по функциональности коллекций достигается повтором кода и агрегированием одного из стандартных механизмов хранения данных.

Основной необходимой функциональностью разрабатываемых коллекций является добавление, удаление, проверка наличия элемента, перебор всех элементов и в большинстве случаев быстрый доступ к элементу по ключу. Подобные механизмы реализуют два класса библиотеки *.NET Framework*: `System.Collections.SortedList` и `System.Collections.Hashtable`. Заметим, что функциональность первого класса шире требуемой. Известно [21, 23, 24], что вторая коллекция значительно быстрее первой, и ее скорость практически не зависит от количества элементов, поэтому основная коллекция библиотеки `LoCoL.Collections.NamedObjectSet` основана (посредством агрегирования) именно на `System.Collections.Hashtable`.

3.4. Сериализация системы автоматов

Для достижения целей, описанных в разд. 1.9. Хранение описания системы автоматов, используем сериализацию и десериализацию объектов.

Сериализация – процесс преобразования экземпляра объекта в последовательность бит, которую можно затем вывести в поток или на носитель.

Десериализация – процесс преобразования сериализованного потока бит в экземпляр некоторого объектного типа. *.NET Framework* предоставляет разработчикам удобный механизм сериализации. Для того чтобы сделать класс, структуру, делегат или перечисление сериализуемым, достаточно снабдить их атрибутом `System.SerializableAttribute`, например, как показано ниже:

```
[Serializable]
class SomeClass
{
          public int m_SomePublicValue = 1000;
          private int m_SomePrivateValue = 1001;
}
```

Так как класс `SomeClass` имеет атрибут `System.SerializableAttribute`, то общеязыковая исполняющая среда (*CLR*) будет выполнять сериализацию автоматически. Для того, чтобы не допустить сериализацию поля, являющегося членом типа с атрибутом `System.SerializableAttribute`, этот член можно снабдить атрибутом `System.NonSerializedAttribute`.

Хотя атрибут `System.SerializableAttribute` прост в использовании, иногда он не соответствует требованиям сериализации. *.NET Framework* позволяет типу с данным атрибутом выполнять свою сериализацию самостоятельно путем реализации интерфейса `System.Runtime.Serialization.ISerializable`, а десериализацию – реализацией специального защищенного конструктора. В библиотеке применяется именно этот механизм для более четкого контроля за сериализуемой

информацией. При этом сохраняется и восстанавливается некоторое дерево объектов, с указанным корневым объектом.

Такое дерево представлено на рис. 2 (здесь и далее используется нотация *UML* [1, 29]). Корневым узлом является объект типа `AutomatonManager`, именно его и рекомендуется сериализовать/десериализовать в коде разработчика. Стоит отметить, что намеренно теряется информация об объектах, содержащих входные переменные, выходные воздействия, протоколирующих объектах, объектах, проверяющих корректность автоматов, так как эта информация не относится непосредственно к системе автоматов, а зависит от конкретного использования ее.

Для сериализации рекомендуется использовать стандартный форматировщик (*Formatter*), предлагаемый библиотекой *.NET Framework* – `System.Runtime.Serialization.Formatters.Soap.SoapFormatter`, который использует гибкий и распространенный протокол *SOAP* (*Simple Object Access Protocol*) [29], основанный на *XML* [12, 20, 32].

Пример сериализации системы автоматов:

```
AutomatonManager manager;
SoapFormatter formatter;
Stream stream;
// ...
formatter.Serialize( stream, manager );
```

Пример десериализации системы автоматов:

```
SoapFormatter formatter;
Stream stream;
// ...
AutomatonManager manager = (AutomatonManager)formatter.Deserialize( stream );
```

Более подробно с механизмом сериализации/десериализации можно ознакомиться в [10, 24].

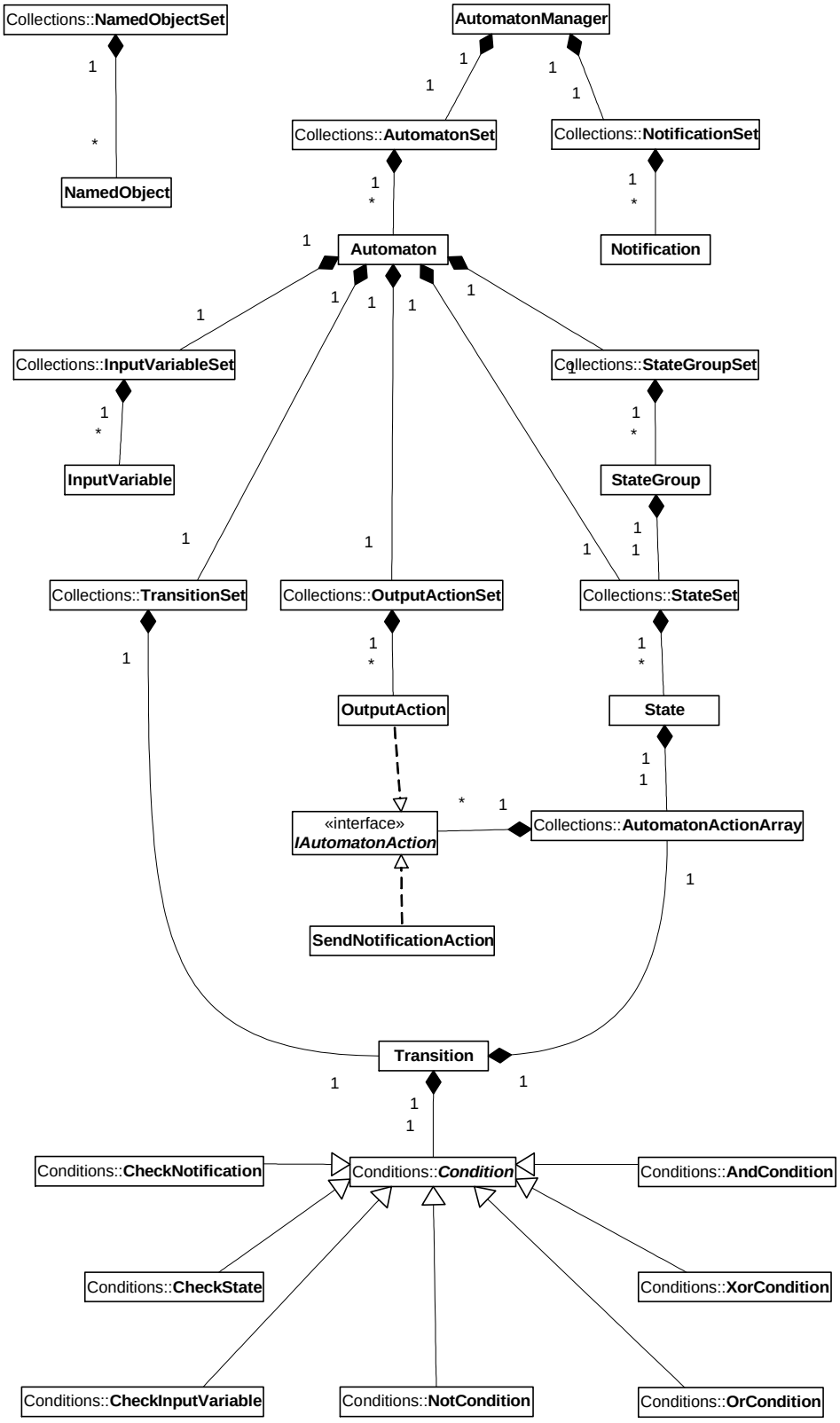

Рис. 2. Диаграмма сериализуемых классов

3.5. Протоколирование, XML и XSLT

Важной частью библиотеки является возможность создания своих механизмов протоколирования работы системы автоматов и настройки его детальности.

Простой реализацией механизма управления детальностью является класс `LoCoL.Logs.DefaultLogDetailer`, позволяющий разрешать/запрещать протоколирование всех событий для определенных автоматов и/или некоторых событий для всех автоматов.

В библиотеке находится две реализации механизма протоколирования:

- `TextWriteLogger`;
- `XmlLogger`.

Первый ведет простой протокол в текстовой форме в стандартный объект библиотеки .NET `System.IO.TextWriter`. Его можно использовать для визуализации протокола во время работы программы. Пример такого протокола:

```
[14:49:58] [ Начало цикла номер 4
[14:49:58] { Автомат A0(Автомат-0) приступил к обработке уведомленияÐ
           m1(Уведомление-1) в состоянии 1(Noname)
[14:49:58] < Автомат A0(Автомат-0) опросил входную переменную x1(ВходнаяÐ
           переменная-1) и получил результат: истина
[14:49:58] T Автомат A0(Автомат-0) совершил переход 1(Noname)[из состоянияÐ
           1(Noname) в состояние 2(Noname)]
[14:49:58]         { Автомат A1(Автомат-1) приступил к обработке уведомленияÐ
           e1(Уведомление-1) в состоянии 0(Noname)
[14:49:58]         N Автомат A1(Автомат-1) сравнил переданное уведомление сÐ
           n1(Уведомление-1) и получил результат: истина
[14:49:58]         T Автомат A1(Автомат-1) совершил переход 0(Noname)[изÐ
           состояния 0(Noname) в состояние 1(Noname)]
[14:49:58]         } Автомат A1(Автомат-1) завершил обработку уведомленияÐ
           e1(Уведомление-1) в состоянии 1(Noname)
[14:49:58] } Автомат A0(Автомат-0) завершил обработку уведомленияÐ
           m1(Уведомление-1)  в состоянии 2(Noname)
[14:49:58] ] Окончание цикла номер 4
```

Второй класс ведет протокол, используя *XML* (*eXtensible Markup Language*, *Расширяемый Язык Разметки*) [12, 20, 32]. Этот выбор легко объясним очень большой распространенностью и гибкостью данного языка. Этот механизм удобно использовать для обработки протоколов внешними программами. Одним

из удобных инструментов преобразования *XML*-протокола является *XSLT* (*eXtensible Stylesheet Language for Transformations*, *Расширяемый Язык Таблиц Стилей для Трансформаций*) [30].

Примеры *XML*-протокола, файла трансформации и получаемого протокола находятся в Приложении 2.

3.6. Классы и интерфейсы библиотеки

Библиотека разделена на пять пространств имен (namespaces):

- `LoCoL` – основная часть библиотеки;
- `LoCoL.Collections` – используемые коллекции;
- `LoCoL.Conditions` – набор классов для динамического создания логических выражений;
- `LoCoL.Exceptions` – исключения (*exceptions*) библиотеки;
- `LoCoL.Logs` – механизмы протоколирования.

Библиотека состоит из 44 классов (в том числе два абстрактных), восьми интерфейсов, 11 делегатов и одного перечисления. Приведем их краткое описание. Более подробное описание всех классов, интерфейсов, делегатов, перечислений, методов, свойств, членов и атрибутов можно найти в прилагаемом к работе файле помощи по библиотеке в формате *CHM* (*Compiled HTML Help File*), созданном автоматически при помощи программы *NDoc.NET* [28] с использованием возможности самодокументирования кода в *Microsoft Visual Studio .NET*.

3.6.1. Пространство имен LoCoL

Классы:

- `Automaton` : `NamedObject`, `IValIDCheckable`, `IChangeable`, `ISerializable`

  Автомат. Единица логического контроля.

- `AutomatonManager` : `IValidCheckable`, `IChangeable`, `ISerializable`

  Менеджер автоматов. Механизм, контролирующий работу системы автоматов и ее взаимодействие с окружающей средой.

- `AutomatonManager.TimerCallbackState`

  Внутренний класс менеджера автоматов. Содержит необходимую информацию для запуска автоматов при автоматическом срабатывании таймера.

- `ChangedEventArgs` : `EventArgs`

  Аргумент события изменения класса, реализующего `IChangeable`. Использование зарезервировано на будущее.

- `InputVariable` : `NamedObject`, `ISerializable`

  Входная переменная автомата.

- `InputVariableAttribute` : `Attribute`

  Атрибут, которым необходимо помечать методы, являющиеся входными переменными автомата.

- `NamedObject` : `ISerializable`

  Именованный объект. Абстрактный класс. Является базовым для многих классов библиотеки.

- `Notification` : `NamedObject`, `ISerializable`

  Уведомление.

- `OutputAction` : `NamedObject`, `IAutomatonAction`, `IValidCheckable`, `ISerializable`

  Выходное воздействие автомата.

- `OutputActionAttribute` : `Attribute`

  Атрибут, которым необходимо помечать методы, являющиеся выходными воздействиями автомата.

- `SendNotificationAction` : `IAutomatonAction`, `IValidCheckable`, `ISerializable`

  Действие автомата: посылка сообщения.

- `State` : `NamedObject`, `IValidCheckable`, `IChangeable`, `ISerializable`

  Состояние автомата.

- `StateAccessibilityAutomatonChecker` : `IAutomatonChecker`

  Класс, проверяющий достижимость всех состояний автомата.

- `StateGroup` : `NamedObject`, `IChangeable`, `ISerializable`

  Группа состояний автомата.

- `Transition` : `NamedObject`, `IValidCheckable`, `IChangeable`, `ISerializable`

  Переход автомата.

Интерфейсы:

- `IAutomatonAction` : `IValidCheckable`

  Интерфейс, который должны реализовать классы, описывающие действия автомата, например выходное воздействие, посылка сообщения, генерирование события.

- `IAutomatonChecker`

  Интерфейс, который должен реализовать класс, проверяющий автомат.

- `IAutomatonInput`

  Интерфейс, который должны реализовывать классы, содержащие входные переменные автомата.

- `IAutomatonOutput`

  Интерфейс, который должны реализовывать классы, содержащие выходные воздействия автомата.

- `IChangeable`

  Интерфейс, указывающий, что класс способен изменять свое состояние и сообщать об этом посредством события.

- `IValidCheckable`

  Интерфейс, указывающий, что класс способен проверять своё состояние и сообщать об ошибке в случае неверного состояния при помощи исключения.

Делегаты:

- `Automaton.AutomatonInputDelegate`

  Внутренний тип делегата входных переменных.

- `Automaton.AutomatonOutputDelegate`

  Внутренний тип делегата выходных воздействий.

- `ChangedEventHandler`

  Делегат события изменения класса.

- Перечисление:

- `NotificationType`

  Тип уведомления. Может принимать значения: *Не задан*, *Событие*, *Сообщение*.

На рис. 3 показана диаграмма классов пространства имен `LoCoL`. Заметим, что не указано наследование от класса `System.Object`, так как от него наследуются все классы, у которых родитель не указан явно. На этом рисунке не показана также и реализация интерфейса

`System.Runtime.Serialization.ISerializable`, так как все классы, реализующие его, указаны отдельно в разд. 3.4.

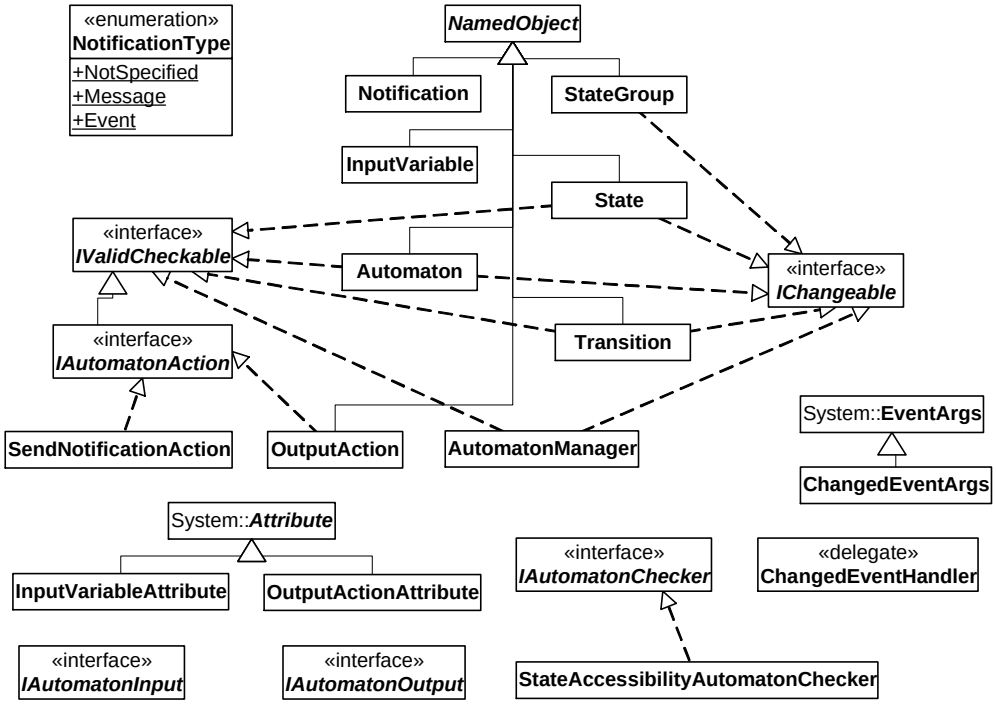

Рис. 3. Диаграмма классов пространства имен `LoCoL`

3.6.2. Пространство имен `LoCoL.Collections`

Классы:

- `AutomatonActionArray : IEnumerable, IChangeable, ISerializable`

  Упорядоченный набор действий автомата.

- `AutomatonSet : IEnumerable, IChangeable, ISerializable`

  Набор автоматов.

- `InputVariableSet : IEnumerable, IChangeable, ISerializable`

  Набор входных переменных.

- `NamedObjectSet : IEnumerable, IChangeable, ISerializable, IDeserializationCallback`

  Набор именованных объектов.

- `NotificationSet : IEnumerable, IChangeable, ISerializable`

  Набор уведомлений.

- `OutputActionSet : IEnumerable, IChangeable, ISerializable`

  Набор выходных воздействий.

- `StateGroupSet : IEnumerable, IChangeable, ISerializable`

  Набор групп состояний.

- `StateSet : IEnumerable, IChangeable, ISerializable`

  Набор состояний.

- `TransitionSet : IEnumerable, IChangeable, ISerializable`

  Набор переходов.

3.6.3. Пространство имен `LoCoL.Conditions`

- `AndCondition : Condition, ISerilaizable`

  Логическое “и".

- `CheckInputVariable : Condition, ISerializable`

  Проверка входной переменной.

- `CheckNotification : Condition, ISerializable`

  Проверка уведомления.

- `CheckState : Condition, ISerializable`

  Проверка состояния автомата в системе.

- `Condition : IEnumerable, IValidCheckable, ISerializable`

  Булево выражение.
- `NotCondition : Condition, ISerializable`

  Логическое отрицание.
- `OrCondition : Condition, ISerializable`

  Логическое “или”.
- `XorCondition : Condition, ISerializable`

  Логическое “исключающее или”.

3.6.4. Пространство имен `LoCoL.Exceptions`

- `AutomatonException : LoCoLException`

  Исключение, генерируемое классом `Automaton`.
- `AutomatonManagerException : LoCoLException`

  Исключение, генерируемое классом `AutomatonManager`.
- `CheckValidException : LoCoLException`

  Исключение, генерируемое при проверки адекватности состояния класса, реализующего интерфейс `IValidCheckable`.
- `ConditionException : LoCoLException`

  Исключение, генерируемое классом `Condition`.
- `LoCoLCollectionsException : LoCoLException`

  Исключение коллекций библиотеки.
- `LoCoLException : ApplicationException`

  Базовый класс всех исключений библиотеки.
- `LoggerException : LoCoLException`

  Исключение, генерируемое классом, реализующим интерфейс `ILogger`.

- `SendNotificationActionException : LoCoLException`

  Исключение, генерируемое классом `SendNotificationAction`.

- `TransitionException : LoCoLException`

  Исключение, генерируемое классом `Transition`.

3.6.5. Пространство имен `LoCoL.Logs`

Классы:

- `DefaultLogDetailer: ILogDetailer`

  Класс, контролирующий детальность протоколирования.

- `TextWriteLogger : ILogger`

  Класс, ведущий протокол в простой текстовой форме.

- `XmlLogger : ILogger`

  Класс, ведущий протокол в форме *XML*-файла.

Интерфейсы:

- `ILogDetailer`

  Интерфейс, который должен реализовать класс, контролирующий детальность протоколирования.

- `ILogger`

  Интерфейс, который должен реализовать класс, протоколирующий работу системы автоматов.

Делегаты:

- `CycleEventHandler`

  Делегат протоколирования начала или окончания элементарного цикла.

- `InputInquireEventHandler`

  Делегат протоколирования опроса входной переменной.

- `NotificationCheckingEventHandler`

  Делегат протоколирования сравнения переданного уведомления.

- `NotificationProcessingEventHandler`

  Делегат протоколирования запуска автомата с некоторым уведомлением или завершения обработки уведомления автоматом.
- `OutputInvocationEventHandler`

  Делегат протоколирования вызова выходного воздействия.
- `SendMessageEventHandler`

  Делегат протоколирования посылки сообщения.
- `StateCheckingEventHandler`

  Делегат протоколирования сравнения состояния некоторого автомата в системе.
- `TransitionEventHandler`

  Делегат протоколирования выполнения перехода.

3.7. Пример использования библиотеки

Использование библиотеки показано на примере системы автоматов, подсчитывающей количество слов в строке и разворачивающей в ней числа. Подробнее задание этого примера описано в [19]. Выбор примера был сделан с целью сравнения с библиотекой *STOOL* [19], о чем речь пойдет далее. Исходные коды приведены в Приложении 3. Графы переходов заимствованы из работы [19] и приведены на рис. 4.

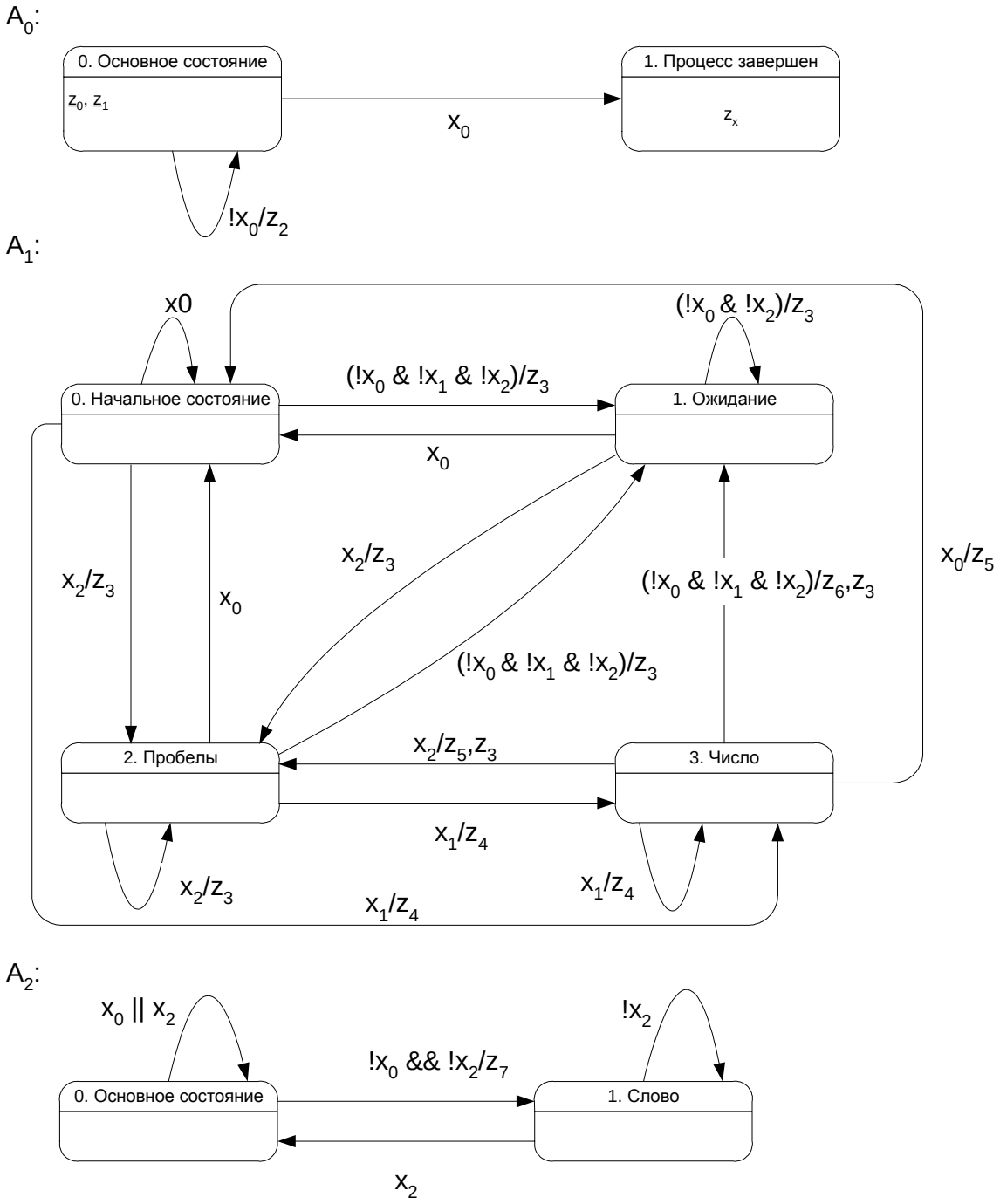

Рис. 4. Графы переходов автоматов

Пример состоит из двух проектов: *LoCoL_Demo_Create* и *LoCoL_Demo*. Первый из них представляет консольное приложение, которое создает систему автоматов и сохраняет ее в файл. Второе приложение загружает систему автоматов из файла, назначает ей объекты с входными переменными и

выходными воздействиями и объект, ведущий протоколирование. Работа системы и ведение протокола визуализируется с использование *Windows-формы*.

3.8. Краткое сравнение с уже существующими библиотеками

Данная работа не является первой попыткой создания библиотеки, совмещающей парадигмы автоматного и объектно-ориентированного программирования. Проведем сравнение с двумя подобными работами: *STOOL* [19] и *Auto-Lib* [12].

Бесспорно, общим у всех проектов является представление автомата как системы взаимосвязанных экземпляров различных классов (например, во всех трех библиотеках существует класс `State`). Все библиотеки спроектированы так, чтобы по возможности вынести в реализацию базовых классов как можно больше кода, который может в последствии дублироваться пользователями. Авторы всех трех работ предлагают обоснованные нововведения, подробный анализ которых читатель может провести самостоятельно.

Данная работа принципиально отличается в вопросе представления информации автомата (например, граф переходов). Если в вышеназванных проектах это достигается написанием кода, то в текущем – созданием набора объектов, что является, очевидно, более динамичным вариантом. Другими словами, автомат представляется не кодом, а данными. Поэтому пользователю не требуется создавать свои классы на основе базовых (например, на основе класса `Auto` из *STOOL*) и переопределять в нем поведение. Отметим также, что в предыдущих работах менее полно использовались возможности ООП [4, 13]. Например, при наследовании автоматов нельзя было дополнить граф переходов новыми состояниями и/или переходами, а лишь полностью изменить его, что ставило под сомнение целесообразность многоуровневой иерархии классов. Динамичность новой библиотеки распространяется практически на все используемые механизмы (например, протоколирование и назначение входных

переменных и выходных воздействий). Значительно улучшена модель взаимодействия автоматов, находящихся в некоторой замкнутой системе. Сериализация системы автоматов является очень удобным инструментом ее переноса между различными приложениями.

Думаю, здесь же стоит упомянуть, что *Microsoft* создает язык *AsmL* (*Abstract State Machine Language*), основанный на теории автоматов [23].

ЗАКЛЮЧЕНИЕ

Был проведен анализ проектов, выполненных в рамках “Движения за открытую проектную документацию” [23]. Были выделены типичные проблемы, такие как:

- необоснованный рост кода автоматных методов;
- недостаточное внимание на взаимосвязь автоматов в системе (в том числе проблемы синхронизации) и связь их с внешней средой;
- необходимость написания кода протоколирования пользователем;
- жесткая привязка входных переменных и выходных воздействий к конкретному автомату на этапе компиляции;
- невозможность динамического изменения структуры автомата (например, добавление состояний и переходов, изменение условий переходов);
- невозможность переноса структуры автоматов между приложениями.

Предложенный подход, заключающийся в динамическом создании и модификации автоматов, позволил решить эти проблемы. Кроме того, было введено различие между синхронными и асинхронными вариантами взаимодействия с автоматами.

Была спроектирована и реализована библиотека *Логическое управление* (*LoCoL – LOgical COntrol Library*), позволяющая достаточно легко и прозрачно применять “автоматное программирование” для программирования поведения объектов с нетривиальной логикой в проектах, создаваемых с использованием объектно-ориентированной парадигмы.

Стоит отметить, что разработанная библиотека успешно применяется в проекте приложения документооборота.

ЛИТЕРАТУРА

1. *Альшевский Ю.А., Раер М.Г., Шалыто А.А.* Механизм обмена сообщениями для параллельно работающих автоматов (на примере системы управления турникетом). СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
2. *Арчер Т.* Основы C#. Новейшие технологии. М.: Русская редакция, 2001.
3. *Астафуров А.А., Шалыто А.А.* Разработка и применения паттерна “Automata”. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
4. *Буч Г.* Объектно-ориентированный анализ и проектирование. М.: Бином, СПб.: Невский диалект, 1998.
5. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. М.: ДМК, 2000.
6. *Гамма Э., Хелм Р., Джонсон Р.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.
7. *Гуисов М.И., Кузнецов А.Б., Шалыто А.А.* Интеграция механизма обмена сообщениями в Switch-технологию. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
8. *Гуисов М.И., Кузнецов А.Б., Шалыто А.А.* Задача Д. Майхилла “Синхронизация цепи стрелков”. Вариант 2. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
9. *Заякин Е.А., Шалыто А.А.* Метод устранения повторных фрагментов кода при реализации конечных автоматов. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.

10. *Маклин С., Нафтел Дж., Уильямс К.* Microsoft .NET Remouting. М.: Русская редакция, 2003.
11. *Наумов А.С., Шалыто А.А.* Система управления лифтом. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
12. *Рэй Э.* Изучаем XML. СПб.: Символ-плюс, 2001.
13. *Страуструп Б.* Язык программирования C++. М.: Бином, СПб.: Невский диалект, 2001.
14. *Фельдман П.И., Шалыто А.А.* Совместное использование объектного и автоматного подходов в программировании. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
15. *Хокканен А.В., Шалыто А.А.* Имитатор игрового автомата класса “Однорукий бандит”. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
16. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
17. *Шалыто А.А., Туккель Н.И.* Танки и автоматы // BYTE/Россия. 2003. № 2. http://is.ifmo.ru/, раздел “Статьи”.
18. *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения “реактивных” систем. // Программирование. 2001. № 5. http://is.ifmo.ru/, раздел “Статьи”.
19. *Шопырин Д.Г., Шалыто А.А.* Объектно-ориентированный подход к автоматному программированию. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru/, раздел “Проекты”.
20. *Эдди С.Э.* XML: Справочник. СПб.: Питер, 1999.

СПИСОК ИНТЕРНЕТ-РЕСУРСОВ

21. *Сайт*, посвященный языку программирования C#. http://www.csharp.net/
22. *Сайт*, посвященный технологии .NET. http://www.gotdotnet.ru/
23. *Сайт* кафедры “Информационные системы” СПбГУ ИТМО. http://is.ifmo.ru/
24. *Microsoft MSDN* Home Page. http://msdn.microsoft.com/
25. *AsmL (Abstract State Machine Language)* Website. http://research.microsoft.com/fse/asml/
26. *Сайт*, посвященный технологии .NET. http://www.opendotnet.ru/
27. *SoftCraft*: разноликое программирование. http://www.softcratf.ru/
28. *NDoc.NET* Home Page. http://ndoc.sourceforge.net/
29. *UML* Resource Page. http://www.uml.org/
30. *W3C XSL*. http://www.w3.org/Style/XSL/
31. *W3C SOAP*. http://www.w3.org/TR/SOAP/
32. *W3C XML*. http://www.w3.org/XML/

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЕ КОДЫ БИБЛИОТЕКИ LOCOL

П1.1. AndCondition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Логическое "И".
    /// </summary>
    [Serializable]
    public class AndCondition : Condition, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        /// <param name="c1">Первый операнд.</param>
        /// <param name="c2">Второй операнд.</param>
        public AndCondition( Condition c1, Condition c2 )
            : base( c1, c2 )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected AndCondition(
            SerializationInfo info,
            StreamingContext context )
            : base( info, context )
        {
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
            SerializationInfo info,
            StreamingContext context )
        {
            base.GetObjectData( info, context );
        }
```

```
        #endregion

        #region Overrided Condition Methods

        /// <summary>
        /// Вычисление выражения "логическое И"
        /// в контексте определенного автомата.
        /// </summary>
        /// <param name="automaton">Автомат, в контексте которого выполняется
        /// вычисление выражения.</param>
        /// <returns>Результат вычисления.</returns>
        public override bool Evaluate( Automaton automaton )
        {
            return
                ((Condition)this[ 0 ]).Evaluate( automaton )
                &&
                ((Condition)this[ 1 ]).Evaluate( automaton );
        }

        #endregion

    }
}
```

П1.2. AssemblyInfo.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System.Reflection;
using System.Runtime.CompilerServices;

[assembly: AssemblyTitle( "LOgical COntrol Library" )]
[assembly: AssemblyCopyright(
    "Copyright (C) by Alexander S. Naumov aka GooRoo, 2004. " +
    "Russia, Saint-Petersburg, SPbIFMO." )]
[assembly: AssemblyVersion( "0.8.*" )]
[assembly: AssemblyDelaySign( false )]
[assembly: AssemblyKeyFile( @"..\..\..\LoCoL_Key.snk" )]
```

П1.3. Automaton.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Collections;
using LoCoL.Exceptions;
using LoCoL.Conditions;
using LoCoL.Logs;
```

```
namespace LoCoL
{
    /// <summary>
    /// Автомат - чистая логика.
    /// </summary>
    [Serializable]
    public class Automaton :
        NamedObject, IValidCheckable, IChangeable, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        /// <param name="id">ИД автомата.</param>
        /// <param name="name">Имя автомата.</param>
        public Automaton( int id, string name )
            : base( id, name )
        {
            m_States = new StateSet();
            m_States.Changed += new ChangedEventHandler( RaiseChangedEvent );
            m_Groups = new StateGroupSet();
            m_Groups.Changed += new ChangedEventHandler( RaiseChangedEvent );
            m_Transitions = new TransitionSet();
            m_Transitions.Changed +=
                new ChangedEventHandler( RaiseChangedEvent );
            m_Transitions.Changed +=
                new ChangedEventHandler( OnTransitionsChanged );
            m_InputVariables = new InputVariableSet();
            m_InputVariables.Changed +=
                new ChangedEventHandler( RaiseChangedEvent );
            m_OutputActions = new OutputActionSet();
            m_OutputActions.Changed +=
                new ChangedEventHandler( RaiseChangedEvent );
            m_AutomatonInputClasses = new ArrayList();
            m_AutomatonInputs = new Hashtable();
            m_AutomatonOutputClasses = new ArrayList();
            m_AutomatonOutputs = new Hashtable();
            m_TransitionsFromState = new Hashtable();
            m_Checkers = new ArrayList();
            Reset();
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected Automaton( SerializationInfo info, StreamingContext context )
            : base( info, context )
        {
            m_States = (StateSet)info.GetValue( "States", typeof(StateSet) );
            m_States.Changed += new ChangedEventHandler( RaiseChangedEvent );
            m_Groups = (StateGroupSet)info.GetValue(
                "Groups", typeof(StateGroupSet) );
            m_Groups.Changed += new ChangedEventHandler( RaiseChangedEvent );
            m_Transitions = (TransitionSet)info.GetValue(
                "Transitions",
```

```
            typeof(TransitionSet) );
        m_Transitions.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_Transitions.Changed +=
            new ChangedEventHandler( OnTransitionsChanged );
        m_InputVariables = (InputVariableSet)info.GetValue(
            "InputVariables",
            typeof(InputVariableSet) );
        m_InputVariables.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_OutputActions = (OutputActionSet)info.GetValue(
            "OutputActions",
            typeof(OutputActionSet) );
        m_OutputActions.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_AutomatonInputClasses = new ArrayList();
        m_AutomatonInputs = new Hashtable();
        m_AutomatonOutputClasses = new ArrayList();
        m_AutomatonOutputs = new Hashtable();
        m_TransitionsFromState = new Hashtable();
        m_Checkers = new ArrayList();
        m_InputVariablesResults = new Hashtable();
        Reset();
    }

    /// <summary>
    /// Контроль над сериализацией объекта.
    /// </summary>
    public override void GetObjectData(
        SerializationInfo info,
        StreamingContext context )
    {
        base.GetObjectData( info, context );
        info.AddValue( "States", m_States );
        info.AddValue( "Groups", m_Groups );
        info.AddValue( "Transitions", m_Transitions );
        info.AddValue( "InputVariables", m_InputVariables );
        info.AddValue( "OutputActions", m_OutputActions );
    }

    #endregion

    #region Private Members

    /// <summary>
    /// Кэширование опроса входных пременных в одном цикле.
    /// </summary>
    private Hashtable m_InputVariablesResults;

    /// <summary>
    /// Кэшируемые переходы из одного состояния.
    /// </summary>
    private Hashtable m_TransitionsFromState;

    /// <summary>
    /// ИД текущего состояния.
    /// </summary>
    private int m_CurrentStateID;

    /// <summary>
    /// Состояния.
```

```
/// </summary>
private StateSet m_States;

/// <summary>
/// Переходы.
/// </summary>
private TransitionSet m_Transitions;

/// <summary>
/// Массив объектов, содержащих входные переменные.
/// </summary>
private ArrayList m_AutomatonInputClasses;

/// <summary>
/// Делегаты входных переменных.
/// </summary>
private Hashtable m_AutomatonInputs;

/// <summary>
/// Массив объектов, содержащих выходные воздействия.
/// </summary>
private ArrayList m_AutomatonOutputClasses;

/// <summary>
/// Делегаты выходных воздействий.
/// </summary>
private Hashtable m_AutomatonOutputs;

/// <summary>
/// Зарегистрированые входные переменные автомата.
/// </summary>
private InputVariableSet m_InputVariables;

/// <summary>
/// Зарегистрированые выходные воздействия автомата.
/// </summary>
private OutputActionSet m_OutputActions;

/// <summary>
/// Группы состояний.
/// </summary>
private StateGroupSet m_Groups;

/// <summary>
/// Менеджер системы автоматов, которой принадлежит текущий автомат.
/// </summary>
private AutomatonManager m_Manager;

/// <summary>
/// Зарегистрированные объекты, проверяющие автомат.
/// </summary>
private ArrayList m_Checkers;

#endregion

#region Private Delegates

/// <summary>
/// Внутренний тип делегата входных переменных.
/// </summary>
private delegate bool AutomatonInputDelegate();
```

```
/// <summary>
/// Внутренний тип делегата выходных воздействий.
/// </summary>
private delegate void AutomatonOutputDelegate();

#endregion

#region Private Methods

/// <summary>
/// Когда изменяются переходы, сбрасывается связанный с ними кэш.
/// </summary>
private void OnTransitionsChanged( object sender, ChangedEventArgs e )
{
    m_TransitionsFromState.Clear();
}

/// <summary>
/// Ловит событие окончания элементарного цикла.
/// Сейчас сбрасывает закэшированные значения входных переменных.
/// </summary>
/// <param name="manger">Не используется.</param>
private void EndCycleEventHandler( AutomatonManager manger )
{
    m_InputVariablesResults.Clear();
}

/// <summary>
/// Возвращает все переходы, ведущие из данного состояния,
/// упорядоченные по убыванию приоритета.
/// </summary>
/// <param name="state">
/// Состояние, для которого определяются переходы.
/// </param>
/// <returns>Все возможные переходы из заданного состояния.</returns>
private Transition[] GetTransitionsFromState( State state )
{
    if ( m_TransitionsFromState[ state.ID ] == null )
    {
        ArrayList array = new ArrayList();
        // Для всех переходов.
        foreach ( Transition t in Transitions )
        {
            // Если переход из состояния и это состояние state,
            // или переход из группы состояний и она содержит state.
            if ( !t.IsFromGroup && t.From == state.ID ||
                t.IsFromGroup &&
                    Groups[ t.From ].States.Contains( state ) )
            {
                // Вставить во вспомогательный массив переход
                // с учетом сортировки по приоритету.
                int index = 0;
                while ( array.Count > index &&
                    ((Transition)array[ index ]).Priority < t.Priority )
                {
                    index++;
                }
                array.Insert( index, t );
            }
        }
```

```
            // Преобразовать в фиксированный массив
            // и запомнить в хеш-таблице.
            m_TransitionsFromState[ state.ID ] =
                (Transition[])array.ToArray( typeof(Transition) );
        }
        // Вернуть закэшированное значение.
        return (Transition[])m_TransitionsFromState[ state.ID ];
    }

    #endregion

    #region Internal Methods

    /// <summary>
    /// Сбрасывает состояние автомата в начальное.
    /// </summary>
    internal void Reset()
    {
        m_CurrentStateID = 0;
    }

    /// <summary>
    /// Выбор перехода.
    /// Вызывается из менеджера автоматов.
    /// </summary>
    /// <returns>
    /// Переход (<see cref="Transition"/>), который автомат
    /// готов выполнить или <c>null</c>,
    /// если переход невозможен.
    /// </returns>
    /// <exception cref="AutomatonException">
    /// Неоднозначность выбора перехода.
    /// </exception>
    internal Transition SelectTransition()
    {
        Transition[] trans =
            GetTransitionsFromState( States[ m_CurrentStateID ] );
        Transition t = null;
        for ( int i = 0; i < trans.Length; i++ )
        {
            // Если пока нет претендентов на искомый переход
            if ( t == null )
            {
                // Если условие перехода выполняется
                if ( trans[ i ].Condition == null ||
                    trans[ i ].Condition.Evaluate( this ) )
                {
                    // Запомнить претендента
                    t = trans[ i ];
                }
                // Перейти к следующему переходу
                continue;
            }
            // Если уже есть претендент и следующий из возможных переходов
            // имееет ниже приоритет, то выбор окончен
            if ( trans[ i ].Priority > t.Priority )
            {
                break;
            }
            // Если есть уже претендент,
            // приоритет его равен приоритету следующего
```

```
            // из возможных и у последнего выполняется условие перехода
            if ( trans[ i ].Condition == null ||
                trans[ i ].Condition.Evaluate( this ) )
            {
                // Сообщить о неоднозначности выбора перехода
                throw new AutomatonException(
                    "Неоднозначность выбора перехода." );
            }
        }
        return t;
    }

    /// <summary>
    /// Проверка входной переменной.
    /// </summary>
    /// <param name="inputVariableID">ИД переменной.</param>
    /// <returns>Результат проверки.</returns>
    internal bool CheckInputVariable( int inputVariableID )
    {
        bool result;
        if ( m_InputVariablesResults[ inputVariableID ] == null )
        {
            InputVariable v = InputVariables[ inputVariableID ];
            Delegate d = (Delegate)m_AutomatonInputs[ inputVariableID ];
            result = (bool)d.DynamicInvoke( null );
            Manager.LogInputInquire( this, v, result );
            m_InputVariablesResults[ inputVariableID ] = result;
        }
        return (bool)m_InputVariablesResults[ inputVariableID ];
    }

    /// <summary>
    /// Выполнение выходного воздействия.
    /// </summary>
    /// <param name="outputActionID">ID воздействия.</param>
    internal void DoOutputInvocation( int outputActionID )
    {
        OutputAction a = OutputActions[ outputActionID ];
        Delegate d = (Delegate)m_AutomatonOutputs[ outputActionID ];
        d.DynamicInvoke( null );
        Manager.LogOutputInvocation( this, a );
    }

    /// <summary>
    /// Совершение перехода.
    /// Вызывается из менеджера автоматов.
    /// </summary>
    /// <param name="transition">
    /// Переход, который автомат должен выполнить.
    /// </param>
    internal void DoTransition( Transition transition )
    {
        State st = States[ transition.To ];
        m_CurrentStateID = transition.To;
        Manager.LogTransition( this, transition );
        transition.Actions.DoActions( this );
        st.Actions.DoActions( this );
    }

    /// <summary>
    /// Устанавливает менеджера автоматов, которому принадлежит
```

```
/// данный автомат. Вызывается из самого менеджера.
/// Устанавливается (и вызывает событие Changed) только в случае,
/// если менеджер новый.
/// </summary>
/// <param name="manager">Устанавливаемый менеджер.</param>
internal void SetManager( AutomatonManager manager )
{
    if ( !ReferenceEquals( m_Manager, manager ) )
    {
        if ( m_Manager != null )
        {
            m_Manager.EndCycleEvent -=
                new CycleEventHandler( EndCycleEventHandler );
        }
        m_Manager = manager;
        m_Manager.EndCycleEvent +=
            new CycleEventHandler( EndCycleEventHandler );
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
}

#endregion

#region Overrided Methods

/// <summary>
/// Строковое описание автомата.
/// </summary>
public override string ToString()
{
    return "A" + base.ToString();
}

#endregion

#region Public Methods

/// <summary>
/// Регистрация объекта, содержащего входные переменные
/// для данного автомата.
/// </summary>
/// <param name="input">Регистрируемый объект.</param>
/// <exception cref="AutomatonException">
/// <list type="bullet">
/// <item><description>
/// У одного из методов класса передаваемого объекта объявлено больше
/// одного атрибута InputVariableAttribute.
/// </description></item>
/// <item><description>
/// Данному автомату назначены два разных метода
/// в качестве одной входной переменной.
/// </description></item>
/// <item><description>
/// Передаваемый объект уже зарегистрирован.
/// </description></item>
/// </list>
/// </exception>
public void RegisterInput( IAutomatonInput input )
{
    if ( m_AutomatonInputClasses.Contains( input ) )
    {
```

```
            throw new AutomatonException(
                "Данный объект уже зарегистрирован." );
        }
        m_AutomatonInputClasses.Add( input );
        Type type = input.GetType();
        foreach ( MethodInfo mi in type.GetMethods() )
        {
            Attribute[] attr = Attribute.GetCustomAttributes(
                mi,
                typeof(InputVariableAttribute) );
            if ( attr.Length == 0 )
            {
                continue;
            }
            if ( attr.Length > 1 )
            {
                throw new AutomatonException( "У метода " +
                    type.FullName + "." + mi.Name +
                    " объявлено больше одного атрибута " +
                    "InputVariableAttribute." );
            }
            int id = ( (InputVariableAttribute)attr[ 0 ] ).ID;
            Delegate d = Delegate.CreateDelegate(
                typeof(AutomatonInputDelegate),
                input,
                mi.Name );
            if ( m_AutomatonInputs[ id ] != null )
            {
                Delegate d0 = (Delegate)m_AutomatonInputs[ id ];
                throw new AutomatonException( "Автомату " +
                    this.ToString() + " назначены два разных метода " +
                    "в качестве одной входной переменной: " +
                    d0.Method.DeclaringType + "." + d0.Method.Name + " и " +
                    d.Method.DeclaringType + "." + d.Method.Name + "." );
            }
            m_AutomatonInputs[ id ] = d;
        }
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }

    /// <summary>
    /// Удаление из списка зарегистрированных объектов,
    /// содержащих входные переменные для данного автомата.
    /// </summary>
    /// <param name="input">Удаляемый объект.</param>
    /// <exception cref="AutomatonException">
    /// Удаляемый объект не был ранее зарегистрирован.
    /// </exception>
    public void UnregisterInput( IAutomatonInput input )
    {
        if ( !m_AutomatonInputClasses.Contains( input ) )
        {
            throw new AutomatonException(
                "Объект не был ранее зарегистрирован." );
        }
        m_AutomatonInputClasses.Remove( input );
        Type type = input.GetType();
        foreach ( MethodInfo mi in type.GetMethods() )
        {
            Attribute[] attr = Attribute.GetCustomAttributes(
                mi,
```

```
            typeof(InputVariableAttribute) );
        int id = ( (InputVariableAttribute)attr[ 0 ] ).ID;
        m_AutomatonInputs.Remove( id );
    }
    RaiseChangedEvent( this, ChangedEventArgs.Empty );
}

/// <summary>
/// Регистрация объекта, содержащего выходные воздействия
/// для данного автомата.
/// </summary>
/// <param name="output">Регистрируемый объект.</param>
/// <exception cref="AutomatonException">
/// <list type="bullet">
/// <item><description>
/// У одного из методов класса передаваемого объекта объявлено больше
/// одного атрибута OutputVariableAttribute.
/// </description></item>
/// <item><description>
/// Данному автомату назначены два разных метода в качестве одного
/// выходного воздействия.
/// </description></item>
/// <item><description>
/// Передаваемый объект уже зарегистрирован.
/// </description></item>
/// </list>
/// </exception>
public void RegisterOutput( IAutomatonOutput output )
{
    if ( m_AutomatonOutputClasses.Contains( output ) )
    {
        throw new AutomatonException(
            "Данный объект уже зарегистрирован." );
    }
    m_AutomatonOutputClasses.Add( output );
    Type type = output.GetType();
    foreach ( MethodInfo mi in type.GetMethods() )
    {
        Attribute[] attr = Attribute.GetCustomAttributes(
            mi,
            typeof(OutputActionAttribute) );
        if ( attr.Length == 0 )
        {
            continue;
        }
        if ( attr.Length > 1 )
        {
            throw new AutomatonException( "У метода "
                + type.FullName + "." + mi.Name +
                " объявлено больше одного атрибута " +
                "OutputVariableAttribute." );
        }
        int id = ( (OutputActionAttribute)attr[ 0 ] ).ID;
        Delegate d = Delegate.CreateDelegate(
            typeof(AutomatonOutputDelegate),
            output,
            mi.Name );
        if ( m_AutomatonOutputs[ id ] != null )
        {
            Delegate d0 = (Delegate)m_AutomatonOutputs[ id ];
            throw new AutomatonException( "Автомату " +
```

```
                this.ToString() + " назначены два разных метода " +
                "в качестве одного выходного воздействия: " +
                d0.Method.DeclaringType + "." + d0.Method.Name + " и " +
                d.Method.DeclaringType + "." + d.Method.Name + "." );
        }
        m_AutomatonOutputs[ id ] = d;
    }
    RaiseChangedEvent( this, ChangedEventArgs.Empty );
}

/// <summary>
/// Удаление из списка зарегистрированных объектов,
/// содержащих выходные воздействия для данного автомата.
/// </summary>
/// <param name="output">Удаляемый объект.</param>
/// <exception cref="AutomatonException">
/// Удаляемый объект не был ранее зарегистрирован.
/// </exception>
public void UnregisterOutput( IAutomatonOutput output )
{
    if ( !m_AutomatonOutputClasses.Contains( output ) )
    {
        throw new AutomatonException(
            "Объект не был ранее зарегистрирован." );
    }
    m_AutomatonOutputClasses.Remove( output );
    Type type = output.GetType();
    foreach ( MethodInfo mi in type.GetMethods() )
    {
        Attribute[] attr = Attribute.GetCustomAttributes(
            mi,
            typeof(OutputActionAttribute) );
        int id = ( (OutputActionAttribute)attr[ 0 ] ).ID;
        m_AutomatonOutputs.Remove( id );
    }
    RaiseChangedEvent( this, ChangedEventArgs.Empty );
}


/// <summary>
/// Регистрация объекта, проверяющего автомат.
/// </summary>
/// <param name="checker">Регистрируемый объект.</param>
/// <exception cref="AutomatonException">
/// Передаваемый объект уже зарегистрирован.
/// </exception>
public void RegisterChecker( IAutomatonChecker checker )
{
    if ( m_Checkers.Contains( checker ) )
    {
        throw new AutomatonException( "Данный объект, " +
            "проверяющий автомат, уже зарегистрирован." );
    }
    m_Checkers.Add( checker );
}

/// <summary>
/// Удаление из списка зарегистрированных объектов, проверяющих автомат.
/// </summary>
/// <param name="checker">Удаляемый объект.</param>
/// <exception cref="AutomatonException">
```

```
/// Удаляемый объект не был ранее зарегистрирован.
/// </exception>
public void UnregisterChecker( IAutomatonChecker checker )
{
    if ( !m_Checkers.Contains( checker ) )
    {
        throw new AutomatonException( "Данный объект, " +
            "проверяющий автомат, не был зарегистрирован." );
    }
    m_Checkers.Remove( checker );
}


#endregion

#region Public Properties

/// <summary>
/// Текущее состояние.
/// </summary>
public State CurrentState
{
    get
    {
        return States[ m_CurrentStateID ];
    }
}

/// <summary>
/// Состояния.
/// </summary>
public StateSet States
{
    get
    {
        return m_States;
    }
}

/// <summary>
/// Группы состояний.
/// </summary>
public StateGroupSet Groups
{
    get
    {
        return m_Groups;
    }
}

/// <summary>
/// Зарегистрированные выходные воздействия автомата.
/// </summary>
public OutputActionSet OutputActions
{
    get
    {
        return m_OutputActions;
    }
}
```

```
/// <summary>
/// Переходы.
/// </summary>
public TransitionSet Transitions
{
    get
    {
        return m_Transitions;
    }
}

/// <summary>
/// Зарегистрированные входные переменные автомата.
/// </summary>
public InputVariableSet InputVariables
{
    get
    {
        return m_InputVariables;
    }
}

/// <summary>
/// Менеджер системы автоматов, которой принадлежит текущий автомат.
/// </summary>
public AutomatonManager Manager
{
    get
    {
        return m_Manager;
    }
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверить адекватность состояния автомата.
/// </summary>
/// <param name="args">В качестве единственного аргумента может быть
/// экземпляр класса, реализующего интерфейс <see cref="IEnumerable"/>,
/// в котором находятся экземпляры классов, реализующих интерфейс
/// <seealso cref="IAutomatonChecker"/>.</param>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Автомат не привязан к системе автоматов.
/// </description></item>
/// <item><description>
/// У автомата отсутствует состояние с ID==0.
/// </description></item>
/// <item><description>
/// У автомата в одной из групп состояний находится состояние,
/// которое отсутствует в самом автомате.
/// </description></item>
/// <item><description>
/// Не назначен метод для одной из входных переменных.
/// </description></item>
/// <item><description>
/// Не назначен метод для одного из выходных воздействий.
```

```
/// </description></item>
/// <item><description>
/// Метод требует единственный параметр типа IEnumerable,
/// в котором находятся экземпляры классов,
/// реализующих интерфейс IAutomatonChecker.
/// </description></item>
/// </list>
/// </exception>
public void CheckValid( params object[] args )
{
    // Проверка привязанности к системе автоматов.
    if ( Manager == null )
    {
        throw new CheckValidException( "Автомат " +
            this.ToString() + " не привязан к системе автоматов." );
    }
    // Проверка всех переходов
    foreach ( Transition t in Transitions )
    {
        t.CheckValid( this );
    }
    // У автомата должно быть состояние с ID==0.
    if ( States[ 0 ] == null )
    {
        throw new CheckValidException( "У автомата " +
            this.ToString() + " отсутствует состояние с ID==0.");
    }
    // Проверка всех состояний
    foreach ( State s in States )
    {
        s.CheckValid( this );
    }
    // Проверка всех групп состояний
    foreach ( StateGroup sg in Groups )
    {
        foreach ( State s in sg.States )
        {
            if ( !this.States.Contains( s ) )
            {
                throw new CheckValidException( "У автомата " +
                    ToString() + " в группе " + sg.ToString() +
                    " находится состояние " + s.ToString() +
                    ", которое отсутствует в самом автомате.");
            }
        }
    }
    // Проверка, что всем зарегистрированным входным переменным
    // назначены методы.
    foreach ( InputVariable iv in InputVariables )
    {
        if ( m_AutomatonInputs[ iv.ID ] == null )
        {
            throw new CheckValidException(
                "Не назначен метод для входной переменной " +
                iv + " у автомата " + ToString() + "." );
        }
    }
    // Проверка, что всем зарегистрированным выходным воздействиям
    // назначены методы.
    foreach ( OutputAction oa in OutputActions )
    {
```

```
            if ( m_AutomatonOutputs[ oa.ID ] == null )
            {
                throw new CheckValidException(
                    "Не назначен метод для выходного воздействия " +
                    oa + " у автомата " + ToString() + "." );
            }
        }
        // Проверка внешними зарегистрированными объектами классов,
        // реализующих IAutomatonChecker.
        foreach ( IAutomatonChecker checker in m_Checkers )
        {
            checker.Check( this );
        }
        // Проверка IAutomatonChecker'ами, переданными первым параметром.
        if ( args.Length == 1 && args[ 0 ] != null )
        {
            if ( !( args[ 0 ] is IEnumerable ) )
            {
                throw new CheckValidException( "Метод " +
                    this.GetType().Name + "." +
                    MethodInfo.GetCurrentMethod().Name +
                    " требует единственный параметр типа IEnumerable, " +
                    "в котором находятся экземпляры классов, реализующих " +
                    "интерфейс IAutomatonChecker. ");
            }
            foreach ( object o in (IEnumerable)args[ 0 ] )
            {
                if ( !( o is IAutomatonChecker ) )
                {
                    throw new CheckValidException( "Метод " +
                        this.GetType().Name + "." +
                        MethodInfo.GetCurrentMethod().Name +
                        " требует единственный параметр типа IEnumerable," +
                        " в котором находятся экземпляры классов, " +
                        "реализующих интерфейс IAutomatonChecker. ");
                }
                ((IAutomatonChecker)o).Check( this );
            }
        }
    }

    #endregion

    #region IChangeable Implementaion

    /// <summary>
    /// Событие, сообщающее, что автомат был изменен.
    /// </summary>
    public event ChangedEventHandler Changed;

    /// <summary>
    /// Посылка сообщения <see cref="Changed"/>.
    /// </summary>
    private void RaiseChangedEvent( object sender, ChangedEventArgs e )
    {
        if ( Changed != null )
        {
            Changed( this, e );
        }
    }
```

```
        #endregion

        #region Operators

        /// <summary>
        /// Перекрытие оператора "<see cref="Automaton"/>==<see cref="State"/>".
        /// Возвращает выражение проверки нахождения автомата <c>a</c>
        /// в состоянии <c>s</c> .
        /// </summary>
        public static Condition operator ==( Automaton a, State s )
        {
            return new CheckState( a.ID, s.ID );
        }

        /// <summary>
        /// Перекрытие оператора "<see cref="Automaton"/>!=<see cref="State"/>".
        /// Возвращает выражение проверки нахождения автомата <c>a</c>
        /// не в состоянии <c>s</c> .
        /// </summary>
        public static Condition operator !=( Automaton a, State s )
        {
            return !( a == s );
        }

        #endregion

        #region Implicit Conversions

        /// <summary>
        /// Имплицитное приведение к типу <see cref="SendNotificationAction"/> -
        /// посылка сообщения <c>e0</c> автомату <c>automaton</c>.
        /// </summary>
        public static implicit operator SendNotificationAction(
            Automaton automaton )
        {
            return new SendNotificationAction(
                0,
                NotificationType.Event,
                automaton.ID );
        }

        #endregion

    }
}
```

П1.4. AutomatonActionArray.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Collections
{
```

```
/// <summary>
/// Упорядоченный набор действий автомата.
/// </summary>
[Serializable]
public class AutomatonActionArray : IEnumerable, IChangeable, ISerializable
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonActionArray()
    {
        m_Array = new ArrayList();
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonActionArray( params IAutomatonAction[] actions )
    {
        m_Array = new ArrayList();
        foreach ( IAutomatonAction a in actions )
        {
            this.Add( a );
        }
    }

    /// <summary>
    /// Конструктор копирования.
    /// </summary>
    /// <param name="actions">Объект, с которого делается копия.</param>
    public AutomatonActionArray( AutomatonActionArray actions )
    {
        m_Array = (ArrayList)actions.m_Array.Clone();
    }

    #endregion

    #region ISerializable Implementation

    /// <summary>
    /// Конструктор десериализации.
    /// </summary>
    protected AutomatonActionArray(
        SerializationInfo info,
        StreamingContext context )
    {
        m_Array = new ArrayList();
        for ( int i = 0; i < info.MemberCount; i++ )
        {
            IAutomatonAction a = (IAutomatonAction)info.GetValue(
                "Value_" + i,
                typeof(IAutomatonAction) );
            this.Add( a );
        }
    }

    /// <summary>
    /// Контроль над сериализацией объекта.
```

```
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    int i = 0;
    foreach ( IAutomatonAction a in m_Array )
    {
        info.AddValue( "Value_" + i, a );
        i++;
    }
}

#endregion

#region Private Members

/// <summary>
/// Внутренний массив, в котором хранятся элементы.
/// </summary>
private ArrayList m_Array;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public AutomatonActionArray Clone()
{
    return new AutomatonActionArray( this );
}

/// <summary>
/// Выполнение всех действий автомата в наборе для автомата
/// <c>automaton</c>.
/// </summary>
/// <param name="automaton">
/// Автомат, для которого выполняются действия.
/// </param>
public void DoActions( Automaton automaton )
{
    foreach ( IAutomatonAction action in this )
    {
        action.DoAction( automaton );
    }
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Array.Clear();
}

/// <summary>
/// Добавить элемент.
/// </summary>
/// <param name="action">Добавляемое действие.</param>
```

```
public int Add( IAutomatonAction action )
{
    int index = m_Array.Add( action );
    RaiseChangedEvent( this, ChangedEventArgs.Empty );
    return index;
}

/// <summary>
/// Проверяет, содержится ли элемент в коллекции.
/// </summary>
/// <param name="action">Проверяемое действие.</param>
public bool Contains( IAutomatonAction action )
{
    return m_Array.Contains( action );
}

/// <summary>
/// Удаляет элемент из коллекции.
/// Если элемент отсутствует, генерируется исключение.
/// </summary>
/// <param name="action">Удаляемое действие.</param>
/// <exception cref="LoCoLCollectionsException">
/// Элемент отсутствует в коллекции.
/// </exception>
public void Remove( IAutomatonAction action )
{
    if ( Contains( action ) )
    {
        m_Array.Remove( action );
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
    else
    {
        throw new LoCoLCollectionsException(
            "Элемент отсутствует в коллекции." );
    }
}

#endregion

#region Public Properties

/// <summary>
/// Стандартный индексатор.
/// </summary>
public IAutomatonAction this[ int index ]
{
    get
    {
        return (IAutomatonAction)m_Array[ index ];
    }
}

/// <summary>
/// Количество элементов.
/// </summary>
public int Count
{
    get
    {
        return m_Array.Count;
```

```
            }
        }

        #endregion

        #region IEnumerable Implementation

        /// <summary>
        /// Enumerator.
        /// </summary>
        public IEnumerator GetEnumerator()
        {
            return m_Array.GetEnumerator();
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что массив действий автомата был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        /// <summary>
        /// Посылка сообщения <see cref="Changed"/>.
        /// </summary>
        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.5. AutomatonManager.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Threading;
using System.IO;
using System.Runtime.Serialization;

using LoCoL.Collections;
using LoCoL.Exceptions;
using LoCoL.Logs;

namespace LoCoL
{

    /// <summary>
```

```
/// Менеджер автоматов.
/// </summary>
[Serializable]
public class AutomatonManager : IValidCheckable, IChangeable, ISerializable
{

    #region Public Constructor

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonManager()
    {
        m_Automatons = new AutomatonSet();
        m_Automatons.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_Automatons.Changed +=
            new ChangedEventHandler( OnAutomatonsChanged );
        m_Notifications = new NotificationSet();
        m_Notifications.Add( new Notification( 0, "<Reserved>" ) );
        m_Notifications.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_Timers = new Hashtable();
        m_MessageQueues = Hashtable.Synchronized( new Hashtable() );
        m_IsCycle = false;
        m_CurrentCycle = -1;
        m_RunnedAutomatons = new AutomatonSet();
        m_AutomatonStates = new Hashtable();
        m_Checkers = new ArrayList();
        m_IsValid = false;
    }

    #endregion

    #region ISerializable Implementation

    /// <summary>
    /// Конструктор десериализации.
    /// </summary>
    protected AutomatonManager(
        SerializationInfo info,
        StreamingContext context )
    {
        m_Automatons = (AutomatonSet)info.GetValue(
            "Automatons",
            typeof(AutomatonSet) );
        m_Automatons.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_Automatons.Changed +=
            new ChangedEventHandler( OnAutomatonsChanged );
        m_Notifications = (NotificationSet)info.GetValue(
            "Notifications",
            typeof(NotificationSet) );
        m_Notifications.Changed +=
            new ChangedEventHandler( RaiseChangedEvent );
        m_Timers = new Hashtable();
        m_MessageQueues = Hashtable.Synchronized( new Hashtable() );
        m_IsCycle = false;
        m_CurrentCycle = -1;
        m_RunnedAutomatons = new AutomatonSet();
        m_AutomatonStates = new Hashtable();
```

```
    m_Checkers = new ArrayList();
    m_IsValid = false;
    m_Title = info.GetString( "Title" );
    m_Description = info.GetString( "Description" );
    m_Copyright = info.GetString( "Copyright" );
    m_Version = (Version)info.GetValue( "Version", typeof(Version) );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Title", m_Title );
    info.AddValue( "Description", m_Description );
    info.AddValue( "Copyright", m_Copyright );
    info.AddValue( "Version", m_Version );
    info.AddValue( "Notifications", m_Notifications );
    info.AddValue( "Automatons", m_Automatons );
}

#endregion

#region Private Members

/// <summary>
/// Валидность состояния. Сбрасывается при изменении системы.
/// </summary>
private bool m_IsValid;

/// <summary>
/// Состояния автоматов на момент начала цикла для тех автоматов,
/// которые его изменили.
/// </summary>
private Hashtable m_AutomatonStates;

/// <summary>
/// Цикл сейчас начат, но не окончен.
/// </summary>
private bool m_IsCycle;

/// <summary>
/// Номер текущего или последнего завершенного цикла.
/// </summary>
private int m_CurrentCycle;

/// <summary>
/// Очереди сообщений для каждого автомата.
/// </summary>
private Hashtable m_MessageQueues;

/// <summary>
/// Запущенные в данном цикле автоматы.
/// </summary>
private AutomatonSet m_RunnedAutomatons;

/// <summary>
/// Текущее обрабатываемое уведомление.
/// </summary>
```

```
private Notification m_CurrentNotification;

/// <summary>
/// Все зарегистрированные в системе таймеры.
/// </summary>
private Hashtable m_Timers;

/// <summary>
/// ID следующего таймера.
/// </summary>
private int m_NextTimerID;

/// <summary>
/// Зарегистрированные объекты, проверяющие все автоматы в системе.
/// </summary>
private ArrayList m_Checkers;

/// <summary>
/// Набор автоматов, принадлежащих системе,
/// контролируемой данным менеджером.
/// </summary>
private AutomatonSet m_Automatons;

/// <summary>
/// Зарегистрированые уведомления системы автоматов.
/// </summary>
private NotificationSet m_Notifications;

/// <summary>
/// Название системы автоматов.
/// </summary>
private string m_Title;

/// <summary>
/// Описание системы автоматов.
/// </summary>
private string m_Description;

/// <summary>
/// Информация о правах.
/// </summary>
private string m_Copyright;

/// <summary>
/// Версия системы автоматов.
/// </summary>
private Version m_Version;

#endregion

#region Private Class

/// <summary>
/// Вспомогательный класс. Передаётся делегату при срабатывании таймера.
/// </summary>
private class TimerCallbackState
{
    /// <summary>
    /// Конструктор.
    /// </summary>
    /// <param name="automatsID">Массив ID автоматов.</param>
```

```
    /// <param name="typeOfNotification">Тип уведомления.</param>
    public TimerCallbackState(
        int[] automatsID,
        NotificationType typeOfNotification )
    {
        if ( automatsID != null )
        {
            AutomatsID = (int[])automatsID.Clone();
        }
        TypeOfNotification = typeOfNotification;
    }
    /// <summary>
    /// Массив ID автоматов.
    /// </summary>
    public int[] AutomatsID;
    /// <summary>
    /// Тип уведомления.
    /// </summary>
    public NotificationType TypeOfNotification;
}

#endregion

#region Private Methods

/// <summary>
/// Возвращает очередь для конкретного автомата в системе.
/// </summary>
/// <param name="automatonID">ID автомата.</param>
/// <returns>Очередь сообщений для заданного автомата.</returns>
/// <exception cref="AutomatonManagerException">
/// Автомат с таким ID отсутствует в системе.
/// </exception>
private Queue MessageQueue( int automatonID )
{
    if ( Automatons[ automatonID ].Equals( null ) )
    {
        throw new AutomatonManagerException(
            "Автомат с таким ID отсутствует в системе." );
    }
    Queue q;
    if ( m_MessageQueues[ automatonID ] == null )
    {
        m_MessageQueues[ automatonID ] =
            q =
            Queue.Synchronized( new Queue() );
    }
    else
    {
        q = (Queue)m_MessageQueues[ automatonID ];
    }
    return q;
}

/// <summary>
/// Запустить автомат с конкретным уведомлением.
/// </summary>
/// <param name="automaton">Запускаемый автомат.</param>
/// <param name="notification">
/// Передаваемое автомату уведомление.
/// </param>
```

```
/// <remarks>
/// Данный метод является безопасным
/// с точки зрения мультипоточности (thread-safe).
/// </remarks>
/// <exception cref="AutomatonManagerException">
/// Нарушение реентерабельности системы автоматов:
/// один из автоматов вызван повторно в одном цикле.
/// </exception>
private void Run( Automaton automaton, Notification notification )
{
    lock ( this )
    {
        // Проверить валидность менеджера.
        CheckValid();
        // Считаем, что автоматически цикл не начат.
        bool isAutoBeginCycle = false;
        // Если цикл не начат.
        if ( !IsCycle )
        {
            // То начинаем его и помечаем,
            // что сделали это автоматически.
            BeginCycle();
            isAutoBeginCycle = true;
        }
        // Проверка реентерабельности.
        if ( m_RunnedAutomatons.Contains( automaton ) )
        {
            throw new AutomatonManagerException(
                "Нарушение реентерабельности системы автоматов: " +
                "Автомат " + automaton.ToString() +
                " вызван повторно в одном цикле." );
        }
        // Добавить автомат в набор уже запущенных в этом цикле.
        m_RunnedAutomatons.Add( automaton );
        // Запомнить текущее обрабатываемое уведомление.
        m_CurrentNotification = notification;
        // Протоколирование начала обработки уведомления.
        LogBeginNotificationProcessing( automaton, notification );
        // Выбрать переход у автомата.
        Transition tr = automaton.SelectTransition();
        // Если таковой имеется.
        if ( tr != null )
        {
            // Сделать переход.
            automaton.DoTransition( tr );
        }
        // Протоколирование окончания обработки уведомления.
        LogEndNotificationProcessing( automaton, notification );
        // Очистить текущее обрабатываемое уведомление.
        m_CurrentNotification = null;
        // Если цикл начинали автоматически.
        if ( isAutoBeginCycle )
        {
            // То стоит его завершить.
            EndCycle();
        }
    }
}

/// <summary>
/// Отлавливает изменения набора автоматов.
```

```
/// Если это изменение самого набора (а не вложенных в него автоматов),
/// то делается проверка, чтобы все автоматы принадлежали менеджеру.
/// </summary>
private void OnAutomatonsChanged( object sender, ChangedEventArgs e )
{
    if ( ReferenceEquals( sender, m_Automatons ) )
    {
        foreach ( Automaton a in m_Automatons )
        {
            a.SetManager( this );
        }
    }
}

/// <summary>
/// Обработка таймера.
/// </summary>
/// <param name="state">Объект типа TimerCallbackState.</param>
private void TimerCallback( object state )
{
    TimerCallbackState st = (TimerCallbackState)state;
    foreach ( int id in st.AutomatsID )
    {
        if ( st.TypeOfNotification == NotificationType.NotSpecified )
        {
            Run( id );
        }
        else
        {
            Run( Automatons[ id ],
                Notifications[ 0 ].CreateNotification(
                    st.TypeOfNotification ) );
        }
    }
}

#endregion

#region Internal Methods

/// <summary>
/// Проверяет, запущен ли текущий автомат с заданным уведомлением.
/// </summary>
/// <param name="notificationID">ID уведомления.</param>
/// <param name="notificationType">Тип уведомления.</param>
/// <returns>Результат проверки.</returns>
/// <exception cref="AutomatonManagerException">
/// В данный момент ни один автомат не обрабатывает уведомление.
/// </exception>
internal bool CheckNotification(
    int notificationID,
    NotificationType notificationType )
{
    if ( m_CurrentNotification == null )
    {
        throw new AutomatonManagerException( "В данный момент " +
            "ни один автомат не обрабатывает уведомление." );
    }
    if ( m_CurrentNotification.ID != notificationID )
    {
        return false;
```

```
    }
    if ( notificationType == NotificationType.NotSpecified ||
        m_CurrentNotification.Type == notificationType )
    {
        return true;
    }
    return false;
}

/// <summary>
/// Проверяет, находится ли заданный автомат в заданном состоянии.
/// Если автомат совершил переход, проверяется состояние,
/// в котором он был на момент начала цикла.
/// </summary>
/// <param name="automatonID">ID автомата.</param>
/// <param name="stateID">ID состояния.</param>
/// <returns>Результат проверки.</returns>
internal bool CheckState( int automatonID, int stateID )
{
    int st;
    if ( m_AutomatonStates[ automatonID ] != null )
    {
        st = (int)m_AutomatonStates[ automatonID ];
    }
    else
    {
        st = m_Automatons[ automatonID ].CurrentState.ID;
    }
    return st == stateID;
}

#endregion

#region Public Methods

/// <summary>
/// Сбросить состояние системы.
/// </summary>
/// <remarks>
/// Сбрасывает состояние всех автоматов.
/// </remarks>
/// <exception cref="AutomatonManagerException">
/// Нельзя сбрасывать состояние системы во время элементарного цикла.
/// </exception>
public void Reset()
{
    if ( m_IsCycle )
    {
        throw new AutomatonManagerException( "Нельзя сбрасывать " +
            "состояние системы во время элементарного цикла." );
    }
    foreach ( Automaton a in Automatons )
    {
        a.Reset();
    }
}

/// <summary>
/// Начать элементарный цикл.
/// </summary>
/// <exception cref="AutomatonManagerException">
```

```
/// Цикл уже начат.
/// </exception>
public void BeginCycle()
{
    if ( m_IsCycle )
    {
        throw new AutomatonManagerException( "Цикл уже начат." );
    }
    m_IsCycle = true;
    m_CurrentCycle++;
    m_RunnedAutomatons.Clear();
    m_AutomatonStates.Clear();
    LogBeginCycle();
}

/// <summary>
/// Завершить элементарный цикл.
/// </summary>
/// <exception cref="AutomatonManagerException">
/// Цикл не был начат.
/// </exception>
public void EndCycle()
{
    if ( !m_IsCycle )
    {
        throw new AutomatonManagerException( "Цикл не был начат." );
    }
    m_IsCycle = false;
    LogEndCycle();
}

/// <summary>
/// Послать сообщение автомату.
/// Уведомление помещается в очередь.
/// </summary>
/// <param name="automatonID">ID автомата.</param>
/// <param name="notificationID">ID уведомления.</param>
/// <exception cref="AutomatonManagerException">
/// Уведомление с таким ID не зарегистрировано в системе.
/// </exception>
public void SendMessage( int automatonID, int notificationID )
{
    if ( Notifications[ notificationID ] == null )
    {
        throw new AutomatonManagerException( "Уведомление с таким ID " +
            "не зарегистрировано в системе." );
                    }
    MessageQueue( automatonID ).Enqueue( notificationID );
    LogSendMessage(
        Automatons[ automatonID ],
        Notifications[ notificationID ].CreateMessage() );
}

/// <summary>
/// Инициировать событие автомату.
/// Вызывается автомат с передаваемым уведомлением.
/// </summary>
/// <param name="automatonID">ID автомата.</param>
/// <param name="notificationID">ID уведомления.</param>
/// <exception cref="AutomatonManagerException">
/// Уведомление с таким ID не зарегистрировано в системе.
```

```
/// </exception>
public void RaiseEvent( int automatonID, int notificationID )
{
    if ( Notifications[ notificationID ] == null )
    {
        throw new AutomatonManagerException( "Уведомление с таким ID " +
            "не зарегистрировано в системе." );
    }
    Run( Automatons[ automatonID ],
        Notifications[ notificationID ].CreateEvent() );
}

/// <summary>
/// Запустить автомат с сообщением из очереди или,
/// если сообщений нет, то с сообщением m0.
/// </summary>
/// <param name="automatonID">ID автомата.</param>
public void Run( int automatonID )
{
    int id =
        ( MessageQueue( automatonID ).Count == 0 ) ?
        ( 0 ) :
        ( (int)MessageQueue( automatonID ).Dequeue() );
    Run( Automatons[ automatonID ],
        Notifications[ id ].CreateMessage() );
}

/// <summary>
/// Регистрация объекта, проверяющего все автоматы в системе.
/// </summary>
/// <exception cref="AutomatonException">
/// Данный объект, проверяющий автомат, уже зарегистрирован.
/// </exception>
public void RegisterChecker( IAutomatonChecker checker )
{
    if ( m_Checkers.Contains( checker ) )
    {
        throw new AutomatonException( "Данный объект, " +
            "проверяющий автомат, уже зарегистрирован." );
    }
    m_Checkers.Add( checker );
}

/// <summary>
/// Удаление из списка зарегистрированных объектов,
/// проверяющих все автоматы в системе.
/// </summary>
/// <exception cref="AutomatonException">
/// Данный объект, проверяющий автомат, не был зарегистрирован.
/// </exception>
public void UnregisterChecker( IAutomatonChecker checker )
{
    if ( !m_Checkers.Contains( checker ) )
    {
        throw new AutomatonException( "Данный объект, " +
            "проверяющий автомат, не был зарегистрирован." );
    }
    m_Checkers.Remove( checker );
}

/// <summary>
```

```
/// Установить таймер на многократную посылку автоматам
/// сообщения <c>m0</c> в едином цикле.
/// </summary>
/// <param name="automatsID">Массив ID автоматов.</param>
/// <param name="millisec">
/// Количество миллисекунд, через которые
/// будет повторяться срабатывание таймера.
/// </param>
/// <returns>ID таймера.</returns>
public int SetMessageTimer( int[] automatsID, int millisec )
{
    Timer timer = new Timer(
        new TimerCallback( TimerCallback ),
        new TimerCallbackState( automatsID, NotificationType.Message ),
        millisec,
        millisec );
    m_NextTimerID++;
    m_Timers[ m_NextTimerID ] = timer;
    return m_NextTimerID;
}

/// <summary>
/// Установить таймер на многократный запуск автоматов с сообщением
/// из очереди (или <c>m0</c>, если очередь пуста).
/// </summary>
/// <param name="automatsID">Массив ID автоматов.</param>
/// <param name="millisec">
/// Количество миллисекунд, через которые будет повторяться
/// срабатывание таймера.
/// </param>
/// <returns>ID таймера.</returns>
public int SetRunTimer( int[] automatsID, int millisec )
{
    Timer timer = new Timer(
        new TimerCallback( TimerCallback ),
        new TimerCallbackState(
            automatsID,
            NotificationType.NotSpecified ),
        millisec,
        millisec );
    m_NextTimerID++;
    m_Timers[ m_NextTimerID ] = timer;
    return m_NextTimerID;
}

/// <summary>
/// Установить таймер на многократный запуск автоматов
/// с событием <c>e0</c> в едином цикле.
/// </summary>
/// <param name="automatsID">Массив ID автоматов.</param>
/// <param name="millisec">
/// Количество миллисекунд, через которые будет повторяться
/// срабатывание таймера.
/// </param>
/// <returns>ID таймера.</returns>
public int SetEventTimer( int[] automatsID, int millisec )
{
    Timer timer = new Timer(
        new TimerCallback( TimerCallback ),
        new TimerCallbackState( automatsID, NotificationType.Event ),
        millisec,
```

```
            millisec );
        m_NextTimerID++;
        m_Timers[ m_NextTimerID ] = timer;
        return m_NextTimerID;
    }

    /// <summary>
    /// Убрать таймер.
    /// </summary>
    /// <param name="timerID">ID таймера.</param>
    /// <exception cref="AutomatonManagerException">
    /// Таймера с таким ID не существует.
    /// </exception>
    public void KillTimer( int timerID )
    {
        if ( !m_Timers.ContainsKey( timerID ) )
        {
            throw new AutomatonManagerException(
                "Таймера с таким ID не существует." );
        }
        Timer timer = (Timer)m_Timers[ timerID ];
        m_Timers.Remove( timerID );
        timer.Dispose();
    }

    /// <summary>
    /// Подписать объект <c>logger</c> на события протоколирования.
    /// </summary>
    /// <param name="logger">Подписываемый объект.</param>
    public void RegisterLogger( ILogger logger )
    {
        BeginCycleEvent +=
            new CycleEventHandler(
                logger.OnBeginCycle );
        BeginNotificationProcessingEvent +=
            new NotificationProcessingEventHandler(
                logger.OnBeginNotificationProcessing );
        InputInquireEvent +=
            new InputInquireEventHandler(
                logger.OnInputInquire );
        NotificationCheckingEvent +=
            new NotificationCheckingEventHandler(
                logger.OnNotificationChecking );
        StateCheckingEvent +=
            new StateCheckingEventHandler(
                logger.OnStateChecking );
        TransitionEvent +=
            new TransitionEventHandler(
                logger.OnTransition );
        OutputInvocationEvent +=
            new OutputInvocationEventHandler(
                logger.OnOutputInvocation );
        SendMessageEvent +=
            new SendMessageEventHandler(
                logger.OnSendMessage );
        EndNotificationProcessingEvent +=
            new NotificationProcessingEventHandler(
                logger.OnEndNotificationProcessing );
        EndCycleEvent +=
            new CycleEventHandler(
                logger.OnEndCycle );
```

```
}

/// <summary>
/// Отписать объект <c>logger</c> от событий протоколирования.
/// </summary>
/// <param name="logger">Отписываемый объект.</param>
public void UnregisterLogger( ILogger logger )
{
    BeginCycleEvent -=
        new CycleEventHandler(
            logger.OnBeginCycle );
    BeginNotificationProcessingEvent -=
        new NotificationProcessingEventHandler(
            logger.OnBeginNotificationProcessing );
    InputInquireEvent -=
        new InputInquireEventHandler(
            logger.OnInputInquire );
    NotificationCheckingEvent -=
        new NotificationCheckingEventHandler(
            logger.OnNotificationChecking );
    StateCheckingEvent -=
        new StateCheckingEventHandler(
            logger.OnStateChecking );
    TransitionEvent -=
        new TransitionEventHandler(
            logger.OnTransition );
    OutputInvocationEvent -=
        new OutputInvocationEventHandler(
            logger.OnOutputInvocation );
    SendMessageEvent -=
        new SendMessageEventHandler(
            logger.OnSendMessage );
    EndNotificationProcessingEvent -=
        new NotificationProcessingEventHandler(
            logger.OnEndNotificationProcessing );
    EndCycleEvent -=
        new CycleEventHandler(
            logger.OnEndCycle );
}

#endregion

#region Public Properties

/// <summary>
/// Цикл сейчас начат, но не окончен.
/// </summary>
public bool IsCycle
{
    get
    {
        return m_IsCycle;
    }
}

/// <summary>
/// Номер текущего или последнего завершенного цикла.
/// </summary>
public int CurrentCycle
{
    get
```

```
    {
        return m_CurrentCycle;
    }
}

/// <summary>
/// Набор автоматов, принадлежащих системе,
/// контролируемой данным менеджером.
/// </summary>
public AutomatonSet Automatons
{
    get
    {
        return m_Automatons;
    }
}

/// <summary>
/// Зарегистрированные уведомления системы автоматов.
/// </summary>
public NotificationSet Notifications
{
    get
    {
        return m_Notifications;
    }
}


/// <summary>
/// Название системы автоматов.
/// </summary>
public string Title
{
    get
    {
        return m_Title;
    }
    set
    {
        m_Title = value;
    }
}

/// <summary>
/// Описание системы автоматов.
/// </summary>
public string Description
{
    get
    {
        return m_Description;
    }
    set
    {
        m_Description = value;
    }
}

/// <summary>
/// Информация о правах.
```

```
/// </summary>
public string Copyright
{
    get
    {
        return m_Copyright;
    }
    set
    {
        m_Copyright = value;
    }
}

/// <summary>
/// Версия системы автоматов.
/// </summary>
public Version Version
{
    get
    {
        return m_Version;
    }
    set
    {
        m_Version = value;
    }
}

#endregion

#region Private and Internal Log Methods

/// <summary>
/// Протоколирование начала элементарного цикла.
/// </summary>
private void LogBeginCycle()
{
    if ( BeginCycleEvent != null )
    {
        BeginCycleEvent( this );
    }
}

/// <summary>
/// Протоколирование запуска автомата с некоторым уведомлением.
/// </summary>
internal void LogBeginNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    if ( BeginNotificationProcessingEvent != null )
    {
        BeginNotificationProcessingEvent( automaton, notification );
    }
}

/// <summary>
/// Протоколирование опроса входной переменной.
/// </summary>
internal void LogInputInquire(
    Automaton automaton,
```

```
    InputVariable variable,
    bool result )
{
    if ( InputInquireEvent != null )
    {
        InputInquireEvent( automaton, variable, result );
    }
}

/// <summary>
/// Протоколирование сравнения переданного уведомления.
/// </summary>
internal void LogNotificationChecking(
    Automaton automaton,
    Notification notification,
    bool result )
{
    if ( NotificationCheckingEvent != null )
    {
        NotificationCheckingEvent( automaton, notification, result );
    }
}

/// <summary>
/// Протоколирование сравнения состояния некоторого автомата в системе.
/// </summary>
internal void LogStateChecking(
    Automaton automaton,
    Automaton checkedAutomaton,
    State state,
    bool result )
{
    if ( StateCheckingEvent != null )
    {
        StateCheckingEvent(
            automaton,
            checkedAutomaton,
            state,
            result );
    }
}

/// <summary>
/// Протоколирование выполнения перехода.
/// </summary>
internal void LogTransition(
    Automaton automaton,
    Transition transition )
{
    if ( TransitionEvent != null )
    {
        TransitionEvent( automaton, transition );
    }
}

/// <summary>
/// Протоколирование вызова выходного воздействия.
/// </summary>
internal void LogOutputInvocation(
    Automaton automaton,
    OutputAction action )
```

```
{
    if ( OutputInvocationEvent != null )
    {
        OutputInvocationEvent( automaton, action );
    }
}

/// <summary>
/// Протоколирование посылки сообщения.
/// </summary>
private void LogSendMessage(
    Automaton targetAutomaton,
    Notification notification )
{
    if ( SendMessageEvent != null )
    {
        SendMessageEvent( targetAutomaton, notification );
    }
}

/// <summary>
/// Протоколирование завершения обработки уведомления автоматом.
/// </summary>
internal void LogEndNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    if ( EndNotificationProcessingEvent != null )
    {
        EndNotificationProcessingEvent( automaton, notification );
    }
}

/// <summary>
/// Протоколирование окончания элементарного цикла.
/// </summary>
private void LogEndCycle()
{
    if ( EndCycleEvent != null )
    {
        EndCycleEvent( this );
    }
}

#endregion

#region Log Events

/// <summary>
/// Протоколирование начала элементарного цикла.
/// </summary>
public event CycleEventHandler BeginCycleEvent;

/// <summary>
/// Протоколирование запуска автомата с некоторым уведомлением.
/// </summary>
public event NotificationProcessingEventHandler
    BeginNotificationProcessingEvent;

/// <summary>
/// Протоколирование опроса входной переменной.
```

```
/// </summary>
public event InputInquireEventHandler InputInquireEvent;

/// <summary>
/// Протоколирование сравнения переданного уведомления.
/// </summary>
public event NotificationCheckingEventHandler NotificationCheckingEvent;

/// <summary>
/// Протоколирование сравнения состояния некоторого автомата в системе.
/// </summary>
public event StateCheckingEventHandler StateCheckingEvent;

/// <summary>
/// Протоколирование выполнения перехода.
/// </summary>
public event TransitionEventHandler TransitionEvent;

/// <summary>
/// Протоколирование вызова выходного воздействия.
/// </summary>
public event OutputInvocationEventHandler OutputInvocationEvent;

/// <summary>
/// Протоколирование посылки сообщения.
/// </summary>
public event SendMessageEventHandler SendMessageEvent;

/// <summary>
/// Протоколирование завершения обработки уведомления автоматом.
/// </summary>
public event NotificationProcessingEventHandler
    EndNotificationProcessingEvent;

/// <summary>
/// Протоколирование окончания элементарного цикла.
/// </summary>
public event CycleEventHandler EndCycleEvent;

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверить адекватность состояния системы автоматов.
/// </summary>
/// <param name="args">Не требует параметров.</param>
public void CheckValid( params object[] args )
{
    if ( m_IsValid )
    {
        return;
    }
    foreach ( Automaton a in Automatons )
    {
        a.CheckValid( m_Checkers );
    }
    m_IsValid = true;
}

#endregion
```

```
        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что менеджер автоматов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        /// <summary>
        /// Посылка сообщения <see cref="Changed"/>.
        /// </summary>
        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            m_IsValid = false;
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.6. AutomatonSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор автоматов.
    /// </summary>
    [Serializable]
    public class AutomatonSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public AutomatonSet()
        {
            m_Set = new NamedObjectSet();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        /// <param name="automatons"></param>
        public AutomatonSet( AutomatonSet automatons )
```

```
{
    m_Set = automatons.m_Set.Clone();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected AutomatonSet(
    SerializationInfo info,
    StreamingContext context )
{
    m_Set = (NamedObjectSet)info.GetValue(
        "Value",
        typeof(NamedObjectSet) );
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public AutomatonSet Clone()
{
    return new AutomatonSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}
```

```
/// <summary>
/// Добавить автомат.
/// </summary>
public void Add( Automaton automaton )
{
    m_Set.Add( automaton );
}

/// <summary>
/// Проверяет, содержится ли автомат в коллекции.
/// </summary>
public bool Contains( Automaton automaton )
{
    return m_Set.Contains( automaton );
}

/// <summary>
/// Удаляет автомат из коллекции.
/// Если автомат отсутствует, генерируется исключение.
/// </summary>
public void Remove( Automaton automaton )
{
    m_Set.Remove( automaton );
}

#endregion

#region Public Properties

/// <summary>
/// Получить автомат по его ID.
/// ID - свойство самого автомата, а не его позиция в коллекции.
/// </summary>
public Automaton this[ int id ]
{
    get
    {
        return (Automaton)m_Set[ id ];
    }
}

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}

#endregion

#region IEnumerable Implementation

/// <summary>
/// Enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
```

```
            return m_Set.GetEnumerator();
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор автоматов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.7. CheckInputVariable.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Проверка входной переменной.
    /// </summary>
    [Serializable]
    public class CheckInputVariable : Condition, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public CheckInputVariable( int inputVariableID )
            : base( inputVariableID )
        {
        }

        #endregion

        #region ISerializable Implementation
```

```
/// <summary>
/// Конструктор десериализации.
/// </summary>
protected CheckInputVariable(
    SerializationInfo info,
    StreamingContext context )
    : base( info, context, typeof(int) )
{
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
}

#endregion

#region Overrided Condition Methods

/// <summary>
/// Проверка входной переменной.
/// </summary>
public override bool Evaluate( Automaton automaton )
{
    return automaton.CheckInputVariable( (int)this[ 0 ] );
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверка.
/// </summary>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </description></item>
/// <item><description>
/// Автомат не содержит входной переменной с некоторым ID.
/// </description></item>
/// </list>
/// </exception>
public override void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
        throw new CheckValidException( "Метод " +
            this.GetType().Name + "." +
            MethodInfo.GetCurrentMethod().Name +
            " требует единственный параметр типа Automaton." );
    }
    Automaton a = (Automaton)args[ 0 ];
    if ( a.InputVariables[ (int)this[ 0 ] ] == null )
    {
```

```
                throw new CheckValidException( "Автомат " + a.ToString() +
                    " не содержит входной переменной с ID==" +
                    this[ 0 ] + "." );
            }
        }

        #endregion

    }
}
```

П1.8. CheckNotification.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Проверка уведомления.
    /// </summary>
    [Serializable]
    public class CheckNotification : Condition, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public CheckNotification( int notificationID )
            : this( notificationID, NotificationType.NotSpecified )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public CheckNotification(
            int notificationID,
            NotificationType notificationType )
            : base( notificationID, notificationType )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
```

```
/// Конструктор десериализации.
/// </summary>
protected CheckNotification(
    SerializationInfo info,
    StreamingContext context )
    : base( info, context, typeof(int), typeof(NotificationType) )
{
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
}

#endregion

#region Overrided Condition Methods

/// <summary>
/// Проверка уведомления, с которым запущен автомат.
/// </summary>
public override bool Evaluate( Automaton automaton )
{
    bool result = automaton.Manager.CheckNotification(
        (int)this[ 0 ],
        (NotificationType)this[ 1 ] );
    automaton.Manager.LogNotificationChecking(
        automaton,
        automaton.Manager.Notifications[ (int)this[ 0 ] ].
            CreateNotification( (NotificationType)this[ 1 ] ),
        result );
    return result;
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверка.
/// </summary>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </description></item>
/// <item><description>
/// Система автоматов, которой принадлежит автомат
/// не содержит уведомления с некоторым ID.
/// </description></item>
/// </list>
/// </exception>
public override void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
```

```
                throw new CheckValidException( "Метод " +
                    this.GetType().Name + "." +
                    MethodInfo.GetCurrentMethod().Name +
                    " требует единственный параметр типа Automaton." );
            }
            Automaton a = (Automaton)args[ 0 ];
            if ( a.Manager.Notifications[ (int)this[ 0 ] ] == null )
            {
                throw new CheckValidException( "Система автоматов, " +
                    "которой принадлежит автомат " + a.ToString() +
                    " не содержит уведомления с ID==" + this[ 0 ] + "." );
            }
        }

        #endregion

    }
}
```

П1.9. CheckState.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Проверка состояния автомата в системе.
    /// </summary>
    [Serializable]
    public class CheckState : Condition, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public CheckState( int automatonID, int stateID )
            : base( automatonID, stateID )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected CheckState( SerializationInfo info, StreamingContext context )
            : base( info, context, typeof(int), typeof(int) )
        {
```

```
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
}

#endregion

#region Overrided Condition Methods

/// <summary>
/// Проверка состояния конкретного автомата в системе.
/// </summary>
public override bool Evaluate( Automaton automaton )
{
    bool result = automaton.Manager.CheckState(
        (int)this[ 0 ],
        (int)this[ 1 ] );
    Automaton checkedAutomaton =
        automaton.Manager.Automatons[ (int)this[ 0 ] ];
    automaton.Manager.LogStateChecking(
        automaton,
        checkedAutomaton,
        checkedAutomaton.States[ (int)this[ 1 ] ],
        result );
    return result;
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверка.
/// </summary>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </description></item>
/// <item><description>
/// Менеджер автоматов не содержит автомат с некоторым ID.
/// </description></item>
/// <item><description>
/// Автомат не содержит состояния с некоторым ID.
/// </description></item>
/// </list>
/// </exception>
public override void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
        throw new CheckValidException( "Метод " +
            this.GetType().Name + "." +
            MethodInfo.GetCurrentMethod().Name +
```

```
                    " требует единственный параметр типа Automaton." );
            }
            Automaton a = (Automaton)args[ 0 ];
            Automaton targetA = a.Manager.Automatons[ (int)this[ 0 ] ];
            if ( targetA.Equals( null ) )
            {
                throw new CheckValidException( "Менеджер автоматов " +
                    "не содержит автомат с ID==" + this[ 0 ] + "." );
            }
            if ( targetA.States[ (int)this[ 1 ] ] == null )
            {
                throw new CheckValidException( "Автомат " + a.ToString() +
                    " не содержит состояния с ID==" + this[ 1 ] + "." );
            }
        }

        #endregion

    }
}
```

П1.10. Condition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Collections;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Conditions
{

    /// <summary>
    /// Булево выражение.
    /// </summary>
    [Serializable]
    public abstract class Condition :
        IEnumerable, IValidCheckable, ISerializable
    {

        #region Protected Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        protected Condition( params object[] nodes )
        {
            m_Nodes =(object[])nodes.Clone();
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
```

```
/// Конструктор десериализации.
/// </summary>
protected Condition( SerializationInfo info, StreamingContext context )
{
    m_Nodes = new object[ info.MemberCount ];
    for ( int i = 0; i < info.MemberCount; i++ )
    {
        m_Nodes[ i ] = info.GetValue( "Node_" + i, typeof(object) );
    }
}

/// <summary>
/// Конструктор десериализации с контролем получаемых типов.
/// </summary>
protected Condition(
    SerializationInfo info,
    StreamingContext context,
    params Type[] types )
{
    m_Nodes = new object[ info.MemberCount ];
    for ( int i = 0; i < info.MemberCount; i++ )
    {
        m_Nodes[ i ] = info.GetValue( "Node_" + i, types[ i ] );
    }
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    int i = 0;
    foreach ( object o in m_Nodes )
    {
        info.AddValue( "Node_" + i, o );
        i++;
    }
}

#endregion

#region Private Members

/// <summary>
/// Узлы выражения.
/// </summary>
private object[] m_Nodes;

#endregion

#region Public Properties

/// <summary>
/// Доступ к узлам выражения через индексатор.
/// </summary>
/// <exception cref="ConditionException">
/// Индекс вышел за пределы узлов выражения.
/// </exception>
public object this[ int index ]
```

```
{
    get
    {
        if ( index < 0 || index >= m_Nodes.Length )
        {
            throw new ConditionException(
                "Индекс вышел за пределы узлов выражения." );
        }
        return m_Nodes[ index ];
    }
}

#endregion

#region Public Methods

/// <summary>
/// Доступ к узлам выражения через enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
    return m_Nodes.GetEnumerator();
}

/// <summary>
/// Вычисление выражения в контексте определенного автомата.
/// </summary>
public abstract bool Evaluate( Automaton automaton );

#endregion

#region Operators

/// <summary>
/// Перегрузка оператора AND.
/// </summary>
public static Condition operator &( Condition c1, Condition c2 )
{
    if ( c1 == null )
    {
        return c2;
    }
    if ( c2 == null )
    {
        return c1;
    }
    return new AndCondition( c1, c2 );
}

/// <summary>
/// Перегрузка оператора OR.
/// </summary>
public static Condition operator |( Condition c1, Condition c2 )
{
    if ( c1 == null )
    {
        return c2;
    }
    if ( c2 == null )
    {
        return c1;
```

```
    }
    return new OrCondition( c1, c2 );
}

/// <summary>
/// Перегрузка оператора XOR.
/// </summary>
public static Condition operator ^( Condition c1, Condition c2 )
{
    if ( c1 == null )
    {
        return c2;
    }
    if ( c2 == null )
    {
        return c1;
    }
    return new XorCondition( c1, c2 );
}

/// <summary>
/// Перегрузка оператора NOT.
/// </summary>
public static Condition operator !( Condition c )
{
    if ( c == null )
    {
        return c;
    }
    return new NotCondition( c );
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверить все узлы дерева выражений.
/// </summary>
/// <param name="args">Требуется Automaton.</param>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </description></item>
/// <item><description>
/// Выражение построено неверно
/// или в узлах не переопределен метод <see cref="CheckValid"/>.
/// </description></item>
/// </list>
/// </exception>
public virtual void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
        throw new CheckValidException( "Метод " +
            this.GetType().Name + "." +
            MethodInfo.GetCurrentMethod().Name +
            " требует единственный параметр типа Automaton." );
    }
    foreach ( object obj in this )
```

```
            {
                if ( obj is IValidCheckable )
                {
                    ((IValidCheckable)obj).CheckValid( args );
                }
                else
                {
                    throw new CheckValidException(
                        "Выражение построено неверно " +
                        "или в узлах не переопределен метод CheckValid().");
                }
            }
        }

        #endregion

    }
}
```

П1.11. DefaultLogDetailer.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;

namespace LoCoL.Logs
{
    /// <summary>
    /// Класс, контролирующий детальность протоколирования.
    /// </summary>
    public class DefaultLogDetailer : ILogDetailer
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public DefaultLogDetailer( bool isDefaultAutomaonEnabled )
        {
            m_Automaton = new Hashtable();
            m_Object = new object();
            m_IsDefaultAutomaonEnabled = isDefaultAutomaonEnabled;
            m_EnabledBeginCycle = true;
            m_EnabledBeginNotificationProcessing = true;
            m_EnabledInputInquire = true;
            m_EnabledNotificationChecking = true;
            m_EnabledStateChecking = true;
            m_EnabledTransition = true;
            m_EnabledOutputInvocation = true;
            m_EnabledSendMessage = true;
            m_EnabledEndNotificationProcessing = true;
            m_EnabledEndCycle = true;
        }

        #endregion
```

```
#region Private Members

/// <summary>
/// Автоматы, протоколирование запрещено/разрешено (в зависимости
/// от значения <see cref="m_IsDefaultAutomaonEnabled"/>).
/// </summary>
private Hashtable m_Automaton;

/// <summary>
/// Фиктивный объект для использования
/// в качестве значения в <see cref="m_Automaton"/>.
/// </summary>
private object m_Object;

/// <summary>
/// Разрешено ли протоколирование автоматов по умолчанию.
/// </summary>
private bool m_IsDefaultAutomaonEnabled;

/// <summary>
/// Проверить/установить разрешение
/// на протоколирование начала элементарного цикла.
/// </summary>
private bool m_EnabledBeginCycle;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// запуска автомата.
/// </summary>
private bool m_EnabledBeginNotificationProcessing;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// опроса входной переменной.
/// </summary>
private bool m_EnabledInputInquire;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// сравнения переданного уведомления.
/// </summary>
private bool m_EnabledNotificationChecking;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// сравнения состояния некоторого автомата в системе.
/// </summary>
private bool m_EnabledStateChecking;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// выполнения перехода.
/// </summary>
private bool m_EnabledTransition;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// вызова выходного воздействия.
/// </summary>
private bool m_EnabledOutputInvocation;
```

```
/// <summary>
/// Проверить/установить разрешение на протоколирование
/// посылки сообщения.
/// </summary>
private bool m_EnabledSendMessage;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// завершения обработки уведомления автоматом.
/// </summary>
private bool m_EnabledEndNotificationProcessing;

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// завершения обработки уведомления автоматом.
/// </summary>
private bool m_EnabledEndCycle;

#endregion

#region Public Methods

/// <summary>
/// Разрешить/запретить протоколирование конкретного автомата.
/// </summary>
public void EnableAutomaton( int automatonNo, bool enable )
{
    if ( IsDefaultAutomaonEnabled == enable )
    {
        m_Automaton.Remove( automatonNo );
    }
    else
    {
        m_Automaton.Add( automatonNo, m_Object );
    }
}

/// <summary>
/// Разрешено ли протоколирование конкретного автомата.
/// </summary>
public bool IsEnabledAutomaton( int automatonNo )
{
    return
        m_Automaton.ContainsKey( automatonNo ) ^
        IsDefaultAutomaonEnabled;
}

#endregion

#region Public Properties

/// <summary>
/// Разрешено ли протоколирование автоматов по умолчанию.
/// </summary>
public bool IsDefaultAutomaonEnabled
{
    get
    {
        return m_IsDefaultAutomaonEnabled;
    }
}
```

```
/// <summary>
/// Проверить/установить разрешение на протоколирование
/// начала элементарного цикла.
/// </summary>
public bool IsEnabledBeginCycle
{
    get
    {
        return m_EnabledBeginCycle;
    }
    set
    {
        m_EnabledBeginCycle = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// запуска автомата.
/// </summary>
public bool IsEnabledBeginNotificationProcessing
{
    get
    {
        return m_EnabledBeginNotificationProcessing;
    }
    set
    {
        m_EnabledBeginNotificationProcessing = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// опроса входной переменной.
/// </summary>
public bool IsEnabledInputInquire
{
    get
    {
        return m_EnabledInputInquire;
    }
    set
    {
        m_EnabledInputInquire = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// сравнения переданного уведомления.
/// </summary>
public bool IsEnabledNotificationChecking
{
    get
    {
        return m_EnabledNotificationChecking;
    }
    set
```

```
    {
        m_EnabledNotificationChecking = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// сравнения состояния некоторого автомата в системе.
/// </summary>
public bool IsEnabledStateChecking
{
    get
    {
        return m_EnabledStateChecking;
    }
    set
    {
        m_EnabledStateChecking = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// выполнения перехода.
/// </summary>
public bool IsEnabledTransition
{
    get
    {
        return m_EnabledTransition;
    }
    set
    {
        m_EnabledTransition = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// вызова выходного воздействия.
/// </summary>
public bool IsEnabledOutputInvocation
{
    get
    {
        return m_EnabledOutputInvocation;
    }
    set
    {
        m_EnabledOutputInvocation = value;
    }
}

/// <summary>
/// Проверить/установить разрешение на протоколирование
/// посылки сообщения.
/// </summary>
public bool IsEnabledSendMessage
{
    get
    {
```

```
            return m_EnabledSendMessage;
        }
        set
        {
            m_EnabledSendMessage = value;
        }
    }

    /// <summary>
    /// Проверить/установить разрешение на протоколирование
    /// завершения обработки уведомления автоматом.
    /// </summary>
    public bool IsEnabledEndNotificationProcessing
    {
        get
        {
            return m_EnabledEndNotificationProcessing;
        }
        set
        {
            m_EnabledEndNotificationProcessing = value;
        }
    }

    /// <summary>
    /// Проверить/установить разрешение на протоколирование
    /// завершения обработки уведомления автоматом.
    /// </summary>
    public bool IsEnabledEndCycleg
    {
        get
        {
            return m_EnabledEndCycle;
        }
        set
        {
            m_EnabledEndCycle = value;
        }
    }

    #endregion

    #region ILogDetailer Implementation

    /// <summary>
    /// Сообщает, разрешено ли протоколирование
    /// начала элементарного цикла.
    /// </summary>
    bool ILogDetailer.IsEnabledBeginCycle( AutomatonManager manager )
    {
        return IsEnabledBeginCycle;
    }

    /// <summary>
    /// Сообщает, разрешено ли протоколирование
    /// запуска автомата с некоторым уведомлением.
    /// </summary>
    bool ILogDetailer.IsEnabledBeginNotificationProcessing(
        Automaton automaton,
        Notification notification )
    {
```

```
    return IsEnabledAutomaton( automaton.ID ) &&
        IsEnabledBeginNotificationProcessing;
}

/// <summary>
/// Сообщает, разрешено ли протоколирование
/// опроса входной переменной.
/// </summary>
bool ILogDetailer.IsEnabledInputInquire(
    Automaton automaton,
    InputVariable variable )
{
    return IsEnabledAutomaton( automaton.ID ) &&
        IsEnabledInputInquire;
}

/// <summary>
/// Сообщает, разрешено ли протоколирование
/// сравнения переданного уведомления.
/// </summary>
bool ILogDetailer.IsEnabledNotificationChecking(
    Automaton automaton,
    Notification notification )
{
    return IsEnabledAutomaton( automaton.ID ) &&
        IsEnabledNotificationChecking;
}

/// <summary>
/// Сообщает, разрешено ли протоколирование
/// сравнения состояния некоторого автомата в системе.
/// </summary>
bool ILogDetailer.IsEnabledStateChecking(
    Automaton automaton,
    Automaton checkedAutomaton,
    State state )
{
    return IsEnabledAutomaton( automaton.ID ) &&
        IsEnabledStateChecking;
}

/// <summary>
/// Сообщает, разрешено ли протоколирование
/// выполнения перехода.
/// </summary>
bool ILogDetailer.IsEnabledTransition(
    Automaton automaton,
    Transition transition )
{
    return IsEnabledAutomaton( automaton.ID ) &&
        IsEnabledTransition;
}

/// <summary>
/// Сообщает, разрешено ли протоколирование
/// вызова выходного воздействия.
/// </summary>
bool ILogDetailer.IsEnabledOutputInvocation(
    Automaton automaton,
    OutputAction action )
{
```

```
            return IsEnabledAutomaton( automaton.ID ) &&
                IsEnabledOutputInvocation;
        }

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// посылки сообщения.
        /// </summary>
        bool ILogDetailer.IsEnabledSendMessage(
            Automaton targetAutomaton,
            Notification notification )
        {
            return IsEnabledAutomaton( targetAutomaton.ID ) &&
                IsEnabledSendMessage;
        }

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// завершения обработки уведомления автоматом.
        /// </summary>
        bool ILogDetailer.IsEnabledEndNotificationProcessing(
            Automaton automaton,
            Notification notification )
        {
            return IsEnabledAutomaton( automaton.ID ) &&
                IsEnabledEndNotificationProcessing;
        }

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// окончания элементарного цикла.
        /// </summary>
        bool ILogDetailer.IsEnabledEndCycle( AutomatonManager manager )
        {
            return IsEnabledEndNotificationProcessing;
        }

        #endregion
    }
}
```

П1.12. Exceptions.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL.Exceptions
{

    /// <summary>
    /// Базовый класс всех исключений библиотеки.
    /// </summary>
    public class LoCoLException : ApplicationException
    {

        #region Public Constructors
```

```
        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLException()
            : base()
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLException( string message )
            : base( message )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLException( string message, Exception innerException )
            : base( message, innerException )
        {
        }

        #endregion

    }

    /// <summary>
    /// Исключение коллекций библиотеки.
    /// </summary>
    public class LoCoLCollectionsException : LoCoLException
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLCollectionsException()
            : base()
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLCollectionsException( string message )
            : base( message )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoCoLCollectionsException(
            string message,
            Exception innerException )
            : base( message, innerException )
        {
        }
```

```
    #endregion

}

/// <summary>
/// Исключение, генерируемое классом Automaton.
/// </summary>
public class AutomatonException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonException( string message )
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonException( string message, Exception innerException )
        : base( message, innerException )
    {
    }

    #endregion

}

/// <summary>
/// Исключение, генерируемое классом AutomatonManager.
/// </summary>
public class AutomatonManagerException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonManagerException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonManagerException( string message )
```

```
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public AutomatonManagerException(
        string message,
        Exception innerException )
        : base( message, innerException )
    {
    }

    #endregion

}

/// <summary>
/// Исключение, генерируемое при проверке адекватности состояния класса,
/// реализующего интерфейс <see cref="LoCoL.IValidCheckable"/>.
/// </summary>
public class CheckValidException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public CheckValidException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public CheckValidException( string message )
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public CheckValidException( string message, Exception innerException )
        : base( message, innerException )
    {
    }

    #endregion

}

/// <summary>
/// Исключение, генерируемое классом Condition.
/// </summary>
public class ConditionException : LoCoLException
{
```

```
    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public ConditionException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public ConditionException( string message )
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public ConditionException( string message, Exception innerException )
        : base( message, innerException )
    {
    }

    #endregion

}

/// <summary>
/// Исключение, генерируемое классом Transition.
/// </summary>
public class TransitionException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public TransitionException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public TransitionException( string message )
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public TransitionException( string message, Exception innerException )
        : base( message, innerException )
    {
    }
```

```
    #endregion

}

/// <summary>
/// Исключение, генерируемое классом SendNotificationAction.
/// </summary>
public class SendNotificationActionException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public SendNotificationActionException()
        : base()
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public SendNotificationActionException( string message )
        : base( message )
    {
    }

    /// <summary>
    /// Конструктор.
    /// </summary>
    public SendNotificationActionException(
        string message,
        Exception innerException )
        : base( message, innerException )
    {
    }

    #endregion

}


/// <summary>
/// Исключение, генерируемое классом, реализующим интерфейс ILogger.
/// </summary>
public class LoggerException : LoCoLException
{

    #region Public Constructors

    /// <summary>
    /// Конструктор.
    /// </summary>
    public LoggerException()
        : base()
    {
    }

    /// <summary>
```

```
        /// Конструктор.
        /// </summary>
        public LoggerException( string message )
            : base( message )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        public LoggerException( string message, Exception innerException )
            : base( message, innerException )
        {
        }

        #endregion

    }
}
```

П1.13. IAutomatonAction.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{

    /// <summary>
    /// Интерфейс, который должны реализовать классы, описывающие действия
    /// автомата, например выходное воздействие, посылка сообщения,
    /// генерирование события.
    /// </summary>
    public interface IAutomatonAction : IValidCheckable
    {

        #region Methods

        /// <summary>
        /// Выполнить действие автомата для автомата automaton.
        /// </summary>
        void DoAction( Automaton automaton );

        #endregion

    }
}
```

П1.14. IAutomatonChecker.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
```

```
namespace LoCoL
{
    /// <summary>
    /// Интерфейс, который должен реализовать класс, проверяющий автомат.
    /// </summary>
    public interface IAutomatonChecker
    {

        #region Methods

        /// <summary>
        /// Проверка автомата. В случае ошибки должно генерироваться исключение.
        /// </summary>
        void Check( Automaton automaton );

        #endregion

    }
}
```

П1.15. IAutomatonInput.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Интерфейс, который должны реализовавывать классы,
    /// содержащие входные переменные автомата.
    /// </summary>
    public interface IAutomatonInput
    {
    }
}
```

П1.16. IAutomatonOutput.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Интерфейс, который должны реализовавывать классы,
    /// содержащие выходные воздействия автомата.
    /// </summary>
    public interface IAutomatonOutput
    {
    }
}
```

П1.17. IChangeable.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Интерфейс, указывающий, что класс способен изменять свое состояние
    /// и сообщать обэтом посредством события.
    /// </summary>
    public interface IChangeable
    {

        #region Events

        /// <summary>
        /// Событие, сообщающее, что класс изменил свое состояние.
        /// </summary>
        event ChangedEventHandler Changed;

        #endregion

    }

    /// <summary>
    /// Делегат события изменения класса.
    /// </summary>
    public delegate void ChangedEventHandler(
        object sender,
        ChangedEventArgs e );

    /// <summary>
    /// Аргументы события изменения класса.
    /// </summary>
    public class ChangedEventArgs : EventArgs
    {

        /// <summary>
        /// Пустое сообщение.
        /// </summary>
        public new static readonly ChangedEventArgs Empty =
            new ChangedEventArgs();

    }

}
```

П1.18. ILogDetailer.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
```

```
namespace LoCoL.Logs
{
    /// <summary>
    /// Интерфейс, который должен реализовать класс,
    /// контролирующий детальность протоколирования.
    /// </summary>
    public interface ILogDetailer
    {

        #region Methods

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// начала элементарного цикла.
        /// </summary>
        bool IsEnabledBeginCycle( AutomatonManager manager );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// запуска автомата с некоторым уведомлением.
        /// </summary>
        bool IsEnabledBeginNotificationProcessing(
            Automaton automaton,
            Notification notification );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// опроса входной переменной.
        /// </summary>
        bool IsEnabledInputInquire(
            Automaton automaton,
            InputVariable variable );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// сравнения переданного уведомления.
        /// </summary>
        bool IsEnabledNotificationChecking(
            Automaton automaton,
            Notification n );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// сравнения состояния некоторого автомата в системе.
        /// </summary>
        bool IsEnabledStateChecking(
            Automaton automaton,
            Automaton checkedAutomaton,
            State state );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// выполнения перехода.
        /// </summary>
        bool IsEnabledTransition( Automaton automaton, Transition transition );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// вызова выходного воздействия.
        /// </summary>
        bool IsEnabledOutputInvocation(
```

```
            Automaton automaton,
            OutputAction action );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// посылки сообщения.
        /// </summary>
        bool IsEnabledSendMessage(
            Automaton targetAutomaton,
            Notification notification );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// завершения обработки уведомления автоматом.
        /// </summary>
        bool IsEnabledEndNotificationProcessing(
            Automaton automaton,
            Notification notification );

        /// <summary>
        /// Сообщает, разрешено ли протоколирование
        /// окончания элементарного цикла.
        /// </summary>
        bool IsEnabledEndCycle( AutomatonManager manager );

        #endregion

    }
}
```

П1.19. ILogger.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL.Logs
{
    /// <summary>
    /// Интерфейс, который должен реализовать класс,
    /// протоколирующий работу системы автоматов.
    /// </summary>
    public interface ILogger
    {

        #region Methods

        /// <summary>
        /// Протоколирование начала элементарного цикла.
        /// </summary>
        void OnBeginCycle( AutomatonManager manager );

        /// <summary>
        /// Протоколирование запуска автомата с некоторым уведомлением.
        /// </summary>
        void OnBeginNotificationProcessing(
            Automaton automaton,
            Notification notification );
```

```
/// <summary>
/// Протоколирование опроса входной переменной.
/// </summary>
void OnInputInquire(
    Automaton automaton,
    InputVariable variable,
    bool result );

/// <summary>
/// Протоколирование сравнения переданного уведомления.
/// </summary>
void OnNotificationChecking(
    Automaton automaton,
    Notification notification,
    bool result );

/// <summary>
/// Протоколирование сравнения состояния некоторого автомата в системе.
/// </summary>
void OnStateChecking(
    Automaton automaton,
    Automaton checkedAutomaton,
    State state,
    bool result );

/// <summary>
/// Протоколирование выполнения перехода.
/// </summary>
void OnTransition( Automaton automaton, Transition transition );

/// <summary>
/// Протоколирование вызова выходного воздействия.
/// </summary>
void OnOutputInvocation( Automaton automaton, OutputAction action );

/// <summary>
/// Протоколирование посылки сообщения.
/// </summary>
void OnSendMessage(
    Automaton targetAutomaton,
    Notification notification );

/// <summary>
/// Протоколирование завершения обработки уведомления автоматом.
/// </summary>
void OnEndNotificationProcessing(
    Automaton automaton,
    Notification notification );

/// <summary>
/// Протоколирование окончания элементарного цикла.
/// </summary>
void OnEndCycle( AutomatonManager manager );

#endregion

#region Properties

/// <summary>
/// Класс, контролирующий детальность лога.
```

```
        /// </summary>
        ILogDetailer LogDetailer
        {
            get;
            set;
        }

        #endregion

    }
}
```

П1.20. InputVariable.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

using LoCoL.Conditions;

namespace LoCoL
{
    /// <summary>
    /// Входная переменная.
    /// </summary>
    [Serializable]
    public class InputVariable : NamedObject, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public InputVariable( int id, string name )
            : base( id, name )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected InputVariable(
            SerializationInfo info,
            StreamingContext context )
            : base( info, context )
        {
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
```

```
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
}

#endregion

#region Overrided Methods

/// <summary>
/// Строковое описание входной переменной.
/// </summary>
public override string ToString()
{
    return "x" + base.ToString();
}

#endregion

#region Operators

/// <summary>
/// Перегрузка оператора AND для насильного приведения к Condition.
/// </summary>
public static Condition operator &( InputVariable v, Condition c )
{
    return (Condition)v & c;
}

/// <summary>
/// Перегрузка оператора OR для насильного приведения к Condition.
/// </summary>
public static Condition operator |( InputVariable v, Condition c )
{
    return (Condition)v | c;
}

/// <summary>
/// Перегрузка оператора XOR для насильного приведения к Condition.
/// </summary>
public static Condition operator ^( InputVariable v, Condition c )
{
    return (Condition)v ^ c;
}

/// <summary>
/// Перегрузка оператора NOT для насильного приведения к Condition.
/// </summary>
public static Condition operator !( InputVariable v )
{
    return !(Condition)v;
}

#endregion

#region Implicit Conversions

/// <summary>
/// Имплицитное приведение к типу <see cref="CheckInputVariable"/>.
/// </summary>
```

```
        public static implicit operator CheckInputVariable(
            InputVariable variable )
        {
            return new CheckInputVariable( variable.ID );
        }

        #endregion

    }
}
```

П1.21. InputVariableAttribute.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Атрибут метода, объявляющий последнего как входную переменную автомата.
    /// </summary>
    public class InputVariableAttribute : Attribute
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public InputVariableAttribute( int id )
        {
            m_ID = id;
        }

        #endregion

        #region Private Members

        /// <summary>
        /// ID входной переменной.
        /// </summary>
        private int m_ID;

        #endregion

        #region Public Properties

        /// <summary>
        /// ID входной переменной.
        /// </summary>
        public int ID
        {
            get
            {
                return m_ID;
            }
```

```
        }

        #endregion

    }
}
```

П1.22. InputVariableSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор входных переменных.
    /// </summary>
    [Serializable]
    public class InputVariableSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public InputVariableSet()
        {
            m_Set = new NamedObjectSet();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        public InputVariableSet( InputVariableSet variables )
        {
            m_Set = variables.m_Set.Clone();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected InputVariableSet(
            SerializationInfo info,
            StreamingContext context )
        {
            m_Set = (NamedObjectSet)info.GetValue(
                "Value",
                typeof(NamedObjectSet) );
```

```
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public InputVariableSet Clone()
{
    return new InputVariableSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
/// Добавление входной переменной.
/// </summary>
public void Add( InputVariable variable )
{
    m_Set.Add( variable );
}

/// <summary>
/// Проверяет, содержится ли входная переменная в коллекции.
/// </summary>
public bool Contains( InputVariable variable )
{
    return m_Set.Contains( variable );
}

/// <summary>
/// Удаляет входную переменную из коллекции.
/// Если входная переменная отсутствует, генерируется исключение.
```

```
/// </summary>
public void Remove( InputVariable variable )
{
    m_Set.Remove( variable );
}

#endregion

#region Public Properties

/// <summary>
/// Получить входную переменную по ее ID.
/// ID - свойство самой входной переменной, а не ее позиция в коллекции.
/// </summary>
public InputVariable this[ int id ]
{
    get
    {
        return (InputVariable)m_Set[ id ];
    }
}

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}

#endregion

#region IEnumerable Implementation

/// <summary>
/// Enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
    return m_Set.GetEnumerator();
}

#endregion

#region IChangeable Implementation

/// <summary>
/// Событие, сообщающее, что набор автоматов был изменен.
/// </summary>
public event ChangedEventHandler Changed;

private void RaiseChangedEvent( object sender, ChangedEventArgs e )
{
    if ( Changed != null )
    {
        Changed( this, e );
    }
}
```

```
        #endregion

    }
}
```

П1.23. IValidCheckable.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{

    /// <summary>
    /// Интерфейс, указывающий, что класс способен проверять своё состояние
    /// и сообщать об ошибке в случае неверного состояния при помощи исключения.
    /// </summary>
    /// <remarks>
    /// Рекомендуется использовать исключение
    /// <see cref="LoCoL.Exceptions.CheckValidException"/>
    /// или производные от него.
    /// </remarks>
    public interface IValidCheckable
    {

        #region Methods

        /// <summary>
        /// Проверить адекватность состояния.
        /// </summary>
        void CheckValid( params object[] args );

        #endregion

    }
}
```

П1.24. LogDelegates.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL.Logs
{

    /// <summary>
    /// Делегат протоколирования начала или окончания элементарного цикла.
    /// </summary>
    public delegate void CycleEventHandler( AutomatonManager manager );

    /// <summary>
    /// Делегат протоколирования запуска автомата с некоторым уведомлением
```

```
    /// или завершения обработки уведомления автоматом.
    /// </summary>
    public delegate void NotificationProcessingEventHandler(
        Automaton automaton,
        Notification notification );

    /// <summary>
    /// Делегат протоколирования опроса входной переменной.
    /// </summary>
    public delegate void InputInquireEventHandler(
        Automaton automaton,
        InputVariable variable,
        bool result );

    /// <summary>
    /// Делегат протоколирования сравнения переданного уведомления.
    /// </summary>
    public delegate void NotificationCheckingEventHandler(
        Automaton automaton,
        Notification notification,
        bool result );

    /// <summary>
    /// Делегат протоколирования сравнения состояния
    /// некоторого автомата в системе.
    /// </summary>
    public delegate void StateCheckingEventHandler(
        Automaton automaton,
        Automaton checkedAutomaton,
        State state,
        bool result );

    /// <summary>
    /// Делегат протоколирования выполнения перехода.
    /// </summary>
    public delegate void TransitionEventHandler(
        Automaton automaton,
        Transition transition );

    /// <summary>
    /// Делегат протоколирования вызова выходного воздействия.
    /// </summary>
    public delegate void OutputInvocationEventHandler(
        Automaton automaton,
        OutputAction action );

    /// <summary>
    /// Делегат протоколирования посылки сообщения.
    /// </summary>
    public delegate void SendMessageEventHandler(
        Automaton targetAutomaton,
        Notification notification );

}
```

П1.25. NamedObject.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004
```

```
using System;
using System.Runtime.Serialization;

namespace LoCoL
{
    /// <summary>
    /// Именованый объект.
    /// Содержит ID и название.
    /// Большинство классов библиотеки наследуется от данного класса.
    /// </summary>
    [Serializable]
    public abstract class NamedObject : ISerializable
    {

        #region Protected Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        protected NamedObject( int id, string name )
        {
            m_ID = id;
            m_Name = name;
            if ( m_Name == null || m_Name == "" )
            {
                m_Name = "Noname";
            }
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected NamedObject(
            SerializationInfo info,
            StreamingContext context )
        {
            m_ID = (int)info.GetValue( "ID", typeof(int) );
            m_Name = (string)info.GetValue( "Name", typeof(string) );
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public virtual void GetObjectData(
            SerializationInfo info,
            StreamingContext context )
        {
            info.AddValue( "ID", m_ID );
            info.AddValue( "Name", m_Name );
        }

        #endregion

        #region Private Members

        /// <summary>
```

```
/// ID.
/// </summary>
private int m_ID;

/// <summary>
/// Название.
/// </summary>
private string m_Name;

#endregion

#region Public Properties

/// <summary>
/// ID.
/// </summary>
public int ID
{
    get
    {
        return m_ID;
    }
}

/// <summary>
/// Название.
/// </summary>
public string Name
{
    get
    {
        return m_Name;
    }
}

#endregion

#region Overrided Methods

/// <summary>
/// Строковое описание именованого объекта в формате ID(Name).
/// </summary>
public override string ToString()
{
    return ID.ToString() + "(" + Name + ")";
}

/// <summary>
/// Сравнение двух именованных объектов.
/// </summary>
public override bool Equals( object obj )
{
    NamedObject namedObj = obj as NamedObject;
    if ( obj == null )
    {
        return false;
    }
    return ( this.m_ID == namedObj.m_ID ) &&
        ( this.m_Name == namedObj.m_Name );
}
```

```
        /// <summary>
        /// Получение Hash-кода именованного объекта.
        /// </summary>
        public override int GetHashCode()
        {
            return m_ID.GetHashCode()*9 ^ m_Name.GetHashCode()*11;
        }

        #endregion

    }
}
```

П1.26. NamedObjectSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор именованых объектов.
    /// </summary>
    [Serializable]
    public class NamedObjectSet :
        IEnumerable, IChangeable, ISerializable, IDeserializationCallback
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public NamedObjectSet()
        {
            m_Values = new Hashtable();
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        public NamedObjectSet( NamedObjectSet namedObjectSet )
        {
            m_Values = (Hashtable)namedObjectSet.m_Values.Clone();
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
```

```
protected NamedObjectSet(
    SerializationInfo info,
    StreamingContext context )
{
    m_DeserializedObjects = new ArrayList();
    for ( int i = 0; i < info.MemberCount; i++ )
    {
        NamedObject o = (NamedObject)info.GetValue(
            "Value_" + i,
            typeof(NamedObject) );
        m_DeserializedObjects.Add( o );
    }
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    int i = 0;
    foreach ( NamedObject o in m_Values.Values )
    {
        info.AddValue( "Value_" + i, o );
        i++;
    }
}

/// <summary>
/// Массив ссылок на скорее всего не полностью десериализованные
/// объекты, лежащие в коллекции.
/// </summary>
private ArrayList m_DeserializedObjects;

#endregion

#region IDeserializationCallback Implementation

/// <summary>
/// Завершение десериализации.
/// </summary>
public void OnDeserialization( object sender )
{
    m_Values = new Hashtable();
    foreach ( NamedObject o in m_DeserializedObjects )
    {
        this.Add( o );
    }
    m_DeserializedObjects = null;
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private Hashtable m_Values;
```

```
#endregion

#region Public Methods

/// <summary>
/// Создать копию.
/// </summary>
public NamedObjectSet Clone()
{
    return new NamedObjectSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Values.Clear();
    RaiseChangedEvent( this, ChangedEventArgs.Empty );
}

/// <summary>
/// Добавить.
/// </summary>
/// <exception cref="LoCoLCollectionsException">
/// Объект уже присутсвует в коллекции.
/// </exception>
public void Add( NamedObject namedObject )
{
    if ( !Contains( namedObject ) )
    {
        m_Values.Add( namedObject.ID, namedObject );
        if ( namedObject is IChangeable )
        {
            ((IChangeable)namedObject).Changed +=
                new ChangedEventHandler( RaiseChangedEvent );
        }
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
    else
    {
        throw new LoCoLCollectionsException( "Объект " + namedObject +
            " уже присутсвует в коллекции." );
    }
}

/// <summary>
/// Проверяет, содержится ли объект в коллекции.
/// </summary>
public bool Contains( NamedObject namedObject )
{
    return m_Values.ContainsValue( namedObject );
}

/// <summary>
/// Удалить объект из коллекции.
/// </summary>
/// <exception cref="LoCoLCollectionsException">
/// Объект в коллекции отсутствует.
/// </exception>
public void Remove( NamedObject namedObject )
```

```
{
    if ( Contains( namedObject ) )
    {
        m_Values.Remove( namedObject.ID );
        if ( namedObject is IChangeable )
        {
            ((IChangeable)namedObject).Changed -=
                new ChangedEventHandler( RaiseChangedEvent );
        }
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
    else
    {
        throw new LoCoLCollectionsException( "Объект " + namedObject +
            " в коллекции отсутствует." );
    }
}

#endregion

#region Public Properties

/// <summary>
/// Получить объект по его ID.
/// ID - свойство самого объекта, а не его позиция в коллекции.
/// </summary>
public NamedObject this[ int id ]
{
    get
    {
        return (NamedObject)m_Values[ id ];
    }
}

/// <summary>
/// Количество элементов в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Values.Count;
    }
}

#endregion

#region IEnumerable Implementation

/// <summary>
/// Enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
    return m_Values.Values.GetEnumerator();
}

#endregion

#region IChangeable Implementation
```

```
        /// <summary>
        /// Событие, сообщающее, что набор объектов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.27. NotCondition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Логическое отрицание.
    /// </summary>
    [Serializable]
    public class NotCondition : Condition, ISerializable
    {

        #region Publci Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public NotCondition( Condition c )
            : base( c )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected NotCondition(
            SerializationInfo info,
            StreamingContext context )
            : base( info, context )
        {
        }
```

```
        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
            SerializationInfo info,
            StreamingContext context )
        {
            base.GetObjectData( info, context );
        }

        #endregion

        #region Overrided Condition Methods

        /// <summary>
        /// Вычисление выражения "логическое отрицание" в контексте
        /// определенного автомата.
        /// </summary>
        public override bool Evaluate( Automaton automaton )
        {
            return !((Condition)this[ 0 ]).Evaluate( automaton );
        }

        #endregion

    }
}
```

П1.28. Notification.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

namespace LoCoL
{
    /// <summary>
    /// Уведомление.
    /// </summary>
    [Serializable]
    public class Notification : NamedObject, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public Notification( int id, string name )
            : this( id, name, NotificationType.NotSpecified )
        {
        }

        #endregion

        #region Private Constructors
```

```
/// <summary>
/// Конструктор.
/// </summary>
private Notification(
    int id,
    string name,
    NotificationType notificationType )
    : this(
        id,
        name,
        notificationType,
        new Notification[
            Enum.GetValues( typeof(NotificationType) ).Length ] )
{
}

/// <summary>
/// Конструктор.
/// </summary>
private Notification(
    int id,
    string name,
    NotificationType notificationType,
    Notification[] allTypes )
    : base( id, name )
{
    m_Type = notificationType;
    m_AllTypes = allTypes;
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected Notification(
    SerializationInfo info,
    StreamingContext context )
    : base( info, context )
{
    m_Type = (NotificationType)info.GetValue(
        "NotificationType",
        typeof(NotificationType) );
    m_AllTypes = new Notification[
        Enum.GetValues( typeof(NotificationType) ).Length ];
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
    info.AddValue( "NotificationType", m_Type );
}

#endregion
```

```
#region Private Members

/// <summary>
/// Тип уведомления.
/// </summary>
private NotificationType m_Type;

/// <summary>
/// Массив всех типов данного уведомления. Создан для кэширования
/// и избежания создания большого количества одинаковых
/// объектов в памяти.
/// </summary>
private Notification[] m_AllTypes;

#endregion

#region Public Properties

/// <summary>
/// Тип уведомления.
/// </summary>
public NotificationType Type
{
    get
    {
        return m_Type;
    }
}

#endregion

#region Public Methods

/// <summary>
/// Создать уведомление определенного типа.
/// </summary>
public Notification CreateNotification(
    NotificationType notificationType )
{
    if ( m_AllTypes[ (int)notificationType ] == null )
    {
        m_AllTypes[ (int)notificationType ] =
            new Notification( this.ID, this.Name, notificationType );
    }
    return m_AllTypes[ (int)notificationType ];
}

/// <summary>
/// Создать сообщение.
/// </summary>
public Notification CreateMessage()
{
    return CreateNotification( NotificationType.Message );
}

/// <summary>
/// Создать событие.
/// </summary>
public Notification CreateEvent()
{
```

```
            return CreateNotification( NotificationType.Event );
        }

        #endregion

        #region Overrided Methods

        /// <summary>
        /// Строковое описание уведомления.
        /// </summary>
        public override string ToString()
        {
            char c;
            switch ( this.Type )
            {
            case NotificationType.Message:
                c = 'm';
                break;
            case NotificationType.Event:
                c = 'e';
                break;
            default:
                c = 'n';
                break;
            }
            return c + base.ToString();
        }

        #endregion

    }
}
```

П1.29. NotificationSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор уведомлений.
    /// </summary>
    [Serializable]
    public class NotificationSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public NotificationSet()
        {
            m_Set = new NamedObjectSet();
```

```
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Конструктор копирования.
/// </summary>
public NotificationSet( NotificationSet notifications )
{
    m_Set = notifications.m_Set.Clone();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected NotificationSet(
    SerializationInfo info,
    StreamingContext context )
{
    m_Set = (NamedObjectSet)info.GetValue(
        "Value",
        typeof(NamedObjectSet) );
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public NotificationSet Clone()
{
    return new NotificationSet( this );
}

/// <summary>
```

```
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
/// Добавление уведомления.
/// </summary>
public void Add( Notification notification )
{
    m_Set.Add( notification );
}

/// <summary>
/// Проверяет, содержится ли уведомление в коллекции.
/// </summary>
public bool Contains( Notification notification )
{
    return m_Set.Contains( notification );
}

/// <summary>
/// Удаляет уведомление из коллекции.
/// Если уведомление отсутствует, генерируется исключение.
/// </summary>
public void Remove( Notification notification )
{
    m_Set.Remove( notification );
}

#endregion

#region Public Properties

/// <summary>
/// Получить уведомление по его ID.
/// ID - свойство самого уведомления, а не его позиция в коллекции.
/// </summary>
public Notification this[ int id ]
{
    get
    {
        return (Notification)m_Set[ id ];
    }
}

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}

#endregion
```

```
        #region IEnumerable Implementation

        /// <summary>
        /// Enumerator.
        /// </summary>
        public IEnumerator GetEnumerator()
        {
            return m_Set.GetEnumerator();
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор автоматов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion
    }
}
```

П1.30. NotificationType.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Тип уведомления.
    /// </summary>
    public enum NotificationType
    {
        /// <summary>
        /// Не заданный.
        /// </summary>
        NotSpecified,
        /// <summary>
        /// Сообщение.
        /// </summary>
        Message,
        /// <summary>
        /// Событие.
        /// </summary>
        Event
    }
}
```

П1.31. OrCondition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

namespace LoCoL.Conditions
{
    /// <summary>
    /// Логическое ИЛИ.
    /// </summary>
    [Serializable]
    public class OrCondition : Condition, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public OrCondition( Condition c1, Condition c2 )
            : base( c1, c2 )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected OrCondition(
            SerializationInfo info,
            StreamingContext context )
            : base( info, context )
        {
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
            SerializationInfo info,
            StreamingContext context )
        {
            base.GetObjectData( info, context );
        }

        #endregion

        #region Overrided Methods

        /// <summary>
        /// Вычисление выражения "логическое ИЛИ" в контексте
        /// определенного автомата.
        /// </summary>
```

```
        public override bool Evaluate( Automaton automaton )
        {
            return ((Condition)this[ 0 ]).Evaluate( automaton ) ||
                ((Condition)this[ 1 ]).Evaluate( automaton );
        }

        #endregion

    }
}
```

П1.32. OutputAction.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL
{
    /// <summary>
    /// Выходное воздействие.
    /// </summary>
    [Serializable]
    public class OutputAction : NamedObject, IAutomatonAction, ISerializable
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public OutputAction( int id, string name )
            : base( id, name )
        {
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected OutputAction(
            SerializationInfo info,
            StreamingContext context )
            : base( info, context )
        {
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
```

```
            SerializationInfo info,
            StreamingContext context )
        {
            base.GetObjectData( info, context );
        }

        #endregion

        #region IAutomatonAction Implementation

        /// <summary>
        /// Выполнить действие автомата (выходное воздействие) для
        /// автомата <c>automaton</c>.
        /// </summary>
        public void DoAction( Automaton automaton )
        {
            automaton.DoOutputInvocation( this.ID );
        }

        #endregion

        #region IValidCheckable Implementation

        /// <summary>
        /// Проверить адекватность состояния перехода.
        /// </summary>
        /// <param name="args">Требуется Automaton.</param>
        /// <exception cref="CheckValidException">
        /// <list>
        /// <item><description>
        /// Метод требует единственный параметр типа <see cref="Automaton"/>.
        /// </description></item>
        /// <item><description>
        /// Автомат не содержит выходного воздействия с некоторым ID.
        /// </description></item>
        /// </list>
        /// </exception>
        public void CheckValid( params object[] args )
        {
            if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
            {
                throw new CheckValidException( "Метод " +
                    this.GetType().Name + "." +
                    MethodInfo.GetCurrentMethod().Name +
                    " требует единственный параметр типа Automaton." );
            }
            Automaton a = (Automaton)args[ 0 ];
            if ( a.OutputActions[ this.ID ] == null )
            {
                throw new CheckValidException( "Автомат " + a.ToString() +
                    " не содержит выходного воздействия с ID==" +
                    this.ID + "." );
            }
        }

        #endregion

    }
}
```

П1.33. OutputActionAttribute.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;

namespace LoCoL
{
    /// <summary>
    /// Атрибут метода, объявляющий этот метод как выходное воздействие
    /// автомата.
    /// </summary>
    public class OutputActionAttribute : Attribute
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public OutputActionAttribute( int id )
        {
            m_ID = id;
        }

        #endregion

        #region Private Members

        /// <summary>
        /// ID выходного воздействия.
        /// </summary>
        private int m_ID;

        #endregion

        #region Public Properties

        /// <summary>
        /// ID выходного воздействия.
        /// </summary>
        public int ID
        {
            get
            {
                return m_ID;
            }
        }

        #endregion

    }
}
```

П1.34. OutputActionSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
```

```
// E-mail:      gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор выходных воздействий.
    /// </summary>
    [Serializable]
    public class OutputActionSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public OutputActionSet()
        {
            m_Set = new NamedObjectSet();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        public OutputActionSet( OutputActionSet actions )
        {
            m_Set = actions.m_Set.Clone();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected OutputActionSet(
            SerializationInfo info,
            StreamingContext context )
        {
            m_Set = (NamedObjectSet)info.GetValue(
                "Value",
                typeof(NamedObjectSet) );
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public virtual void GetObjectData(
            SerializationInfo info,
            StreamingContext context )
        {
            info.AddValue( "Value", m_Set );
```

```
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public OutputActionSet Clone()
{
    return new OutputActionSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
/// Добавление выходного воздействия.
/// </summary>
public void Add( OutputAction action )
{
    m_Set.Add( action );
}

/// <summary>
/// Проверяет, содержится ли выходное воздействие в коллекции.
/// </summary>
public bool Contains( OutputAction action )
{
    return m_Set.Contains( action );
}

/// <summary>
/// Удаляет выходное воздействие из коллекции.
/// Если выходное воздействие отсутствует, генерируется исключение.
/// </summary>
public void Remove( OutputAction action )
{
    m_Set.Remove( action );
}

#endregion

#region Public Properties

/// <summary>
```

```
        /// Получить выходное воздействие по его ID.
        /// ID - свойство самого выходного воздействия,
        /// а не его позиция в коллекции.
        /// </summary>
        public OutputAction this[ int id ]
        {
            get
            {
                return (OutputAction)m_Set[ id ];
            }
        }

        /// <summary>
        /// Количество состояний в коллекции.
        /// </summary>
        public int Count
        {
            get
            {
                return m_Set.Count;
            }
        }

        #endregion

        #region IEnumerable Implementation

        /// <summary>
        /// Enumerator.
        /// </summary>
        public IEnumerator GetEnumerator()
        {
            return m_Set.GetEnumerator();
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор автоматов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.35. SendNotificationAction.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
```

```
// E-mail:      gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Exceptions;

namespace LoCoL
{
    /// <summary>
    /// Действие автомата: посылка уведомления.
    /// </summary>
    [Serializable]
    public class SendNotificationAction : IAutomatonAction, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public SendNotificationAction(
            Notification notification,
            int toAutomatonID )
            : this( notification.ID, notification.Type, toAutomatonID )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        /// <exception cref="SendNotificationActionException">
        /// Уведомление должно быть типизировано.
        /// </exception>
        public SendNotificationAction(
            int notificationID,
            NotificationType notificationType,
            int toAutomatonID )
        {
            if ( notificationType == NotificationType.NotSpecified )
            {
                throw new SendNotificationActionException(
                    "Уведомление должно быть типизировано." );
            }
            m_NotificationID = notificationID;
            m_NotificationType = notificationType;
            m_ToAutomatonID = toAutomatonID;
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected SendNotificationAction(
            SerializationInfo info,
            StreamingContext context )
```

```
{
    m_NotificationID = (int)info.GetValue(
        "NotificationID",
        typeof(int) );
    m_NotificationType = (NotificationType)info.GetValue(
        "NotificationType",
        typeof(NotificationType) );
    m_ToAutomatonID = (int)info.GetValue(
        "ToAutomatonID",
        typeof(int) );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "NotificationID", m_NotificationID );
    info.AddValue( "NotificationType", m_NotificationType );
    info.AddValue( "ToAutomatonID", m_ToAutomatonID );
}

#endregion

#region Private Members

/// <summary>
/// ID посылаемого уведомления.
/// </summary>
private int m_NotificationID;

/// <summary>
/// Тип посылаемого уведомления.
/// </summary>
private NotificationType m_NotificationType;

/// <summary>
/// ID автомата, которому посылается уведомление.
/// </summary>
private int m_ToAutomatonID;

#endregion

#region IAutomatonAction Implementation

/// <summary>
/// Выполнить действие автомата (посылка уведомления)
/// для автомата <c>automaton</c>.
/// </summary>
public void DoAction( Automaton automaton )
{
    if ( m_NotificationType == NotificationType.Event )
    {
        automaton.Manager.RaiseEvent(
            m_ToAutomatonID,
            m_NotificationID );
    }
    else if ( m_NotificationType == NotificationType.Message )
    {
```

```
                automaton.Manager.SendMessage(
                    m_ToAutomatonID,
                    m_NotificationID );
            }
        }

        #endregion

        #region IValidCheckable Implementation

        /// <summary>
        /// Проверить адекватность действия автомата (посылки уведомления).
        /// </summary>
        /// <param name="args">Требуется Automaton.</param>
        /// <exception cref="CheckValidException">
        /// <list>
        /// <item><description>
        /// Метод требует единственный параметр типа <see cref="Automaton"/>.
        /// </description></item>
        /// <item><description>
        /// В менеджере автоматов, которому принадлежит автомат,
        /// не зарегистрировано уведомление с некоторым ID.
        /// </description></item>
        /// </list>
        /// </exception>
        public void CheckValid( params object[] args )
        {
            if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
            {
                throw new CheckValidException( "Метод " +
                    this.GetType().Name + "." +
                    MethodInfo.GetCurrentMethod().Name +
                    " требует единственный параметр типа Automaton." );
            }
            Automaton a = (Automaton)args[ 0 ];
            if ( a.Manager.Notifications[ m_NotificationID ] == null )
            {
                throw new CheckValidException( "В менеджере автоматов, " +
                    "которому принадлежит " + a.ToString() +
                    ", не зарегистрировано уведомление с ID==" +
                    m_NotificationID.ToString() + "." );
            }
        }

        #endregion

    }
}
```

П1.36. State.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Collections;
```

```
using LoCoL.Exceptions;

namespace LoCoL
{
    /// <summary>
    /// Состояние автомата.
    /// </summary>
    [Serializable]
    public class State :
        NamedObject, IValidCheckable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        /// <param name="id">ID состояния.</param>
        /// <param name="name">Имя состояния.</param>
        public State( int id, string name )
            : this( id, name, null )
        {
        }

        /// <summary>
        /// Конструктор.
        /// </summary>
        /// <param name="id">ID состояния.</param>
        /// <param name="name">Имя состояния.</param>
        /// <param name="actions">
        /// Действия, выполняемые при переходе в состояние.
        /// </param>
        public State( int id, string name, AutomatonActionArray actions )
            : base( id, name )
        {
            m_Actions =
                ( actions == null ) ?
                ( new AutomatonActionArray() ) :
                ( actions.Clone() );
            m_Actions.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        #endregion

        #region ISerializable Implementation

        /// <summary>
        /// Конструктор десериализации.
        /// </summary>
        protected State( SerializationInfo info, StreamingContext context )
            : base( info, context )
        {
            m_Actions = (AutomatonActionArray)info.GetValue(
                "Actions",
                typeof(AutomatonActionArray) );
        }

        /// <summary>
        /// Контроль над сериализацией объекта.
        /// </summary>
        public override void GetObjectData(
```

```
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
    info.AddValue( "Actions", m_Actions );
}

#endregion


#region Private Members

/// <summary>
/// Действия, выполняемые при переходе в состояние.
/// </summary>
private AutomatonActionArray m_Actions;

#endregion

#region Public Properties

/// <summary>
/// Действия, выполняемые при переходе в состояние.
/// </summary>
public AutomatonActionArray Actions
{
    get
    {
        return m_Actions;
    }
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверить адекватность состояния.
/// </summary>
/// <param name="args">Требуется Automaton.</param>
/// <exception cref="CheckValidException">
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </exception>
public void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
        throw new CheckValidException( "Метод " +
            this.GetType().Name + "." +
            MethodInfo.GetCurrentMethod().Name +
            " требует единственный параметр типа Automaton." );
    }
    Automaton a = (Automaton)args[ 0 ];
    foreach ( IAutomatonAction aa in Actions )
    {
        aa.CheckValid( a );
    }
}

#endregion
```

```
        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор действий автомата был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.37. StateAccessibilityAutomatonChecker.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;

using LoCoL.Collections;
using LoCoL.Exceptions;

namespace LoCoL
{
    /// <summary>
    /// Класс, проверяющий достижимость всех состояний автомата.
    /// </summary>
    public class StateAccessibilityAutomatonChecker : IAutomatonChecker
    {

        #region IAutomatonChecker Implementation

        /// <summary>
        /// Проверка автомата.
        /// </summary>
        /// <remarks>
        /// Данный метод является безопасным
        /// с точки зрения мультипоточности (thread-safe).
        /// </remarks>
        /// <exception cref="CheckValidException">
        /// В автомате обнаружены недостижимые состояния.
        /// </exception>
        public void Check( Automaton automaton )
        {
            lock ( this )
            {
                StateSet states = new StateSet( automaton.States );
                m_Transitions = CreateTransitions( automaton );
                Walk( automaton, states, automaton.States[ 0 ] );
                if ( states.Count > 0 )
```

```
        {
            string s = null;
            foreach ( State st in states )
            {
                s += ", " + st.ToString();
            }
            throw new CheckValidException( "В автомате " +
                automaton.ToString() + " обнаружены недостижимые " +
                "состояния: " + s.Substring( 2 ) + "." );
        }
    }
}

#endregion

#region Private Members

/// <summary>
/// Все переходы из конкретного состояния.
/// </summary>
private Hashtable m_Transitions;

#endregion

#region Private Methods

/// <summary>
/// Добавить в <c>hash</c>, что из состояния <c>st</c>
/// есть переход <c>t</c>.
/// </summary>
private void AddTransition( Hashtable hash, State st, Transition t )
{
    if ( hash[ st ] == null )
    {
        hash[ st ] = new ArrayList();
    }
    ArrayList arr = (ArrayList)hash[ st ];
    arr.Add( t );
}

/// <summary>
/// Создать таблицу переходов для автомата <c>automaton</c>.
/// </summary>
/// <returns>
/// Hashtable с наборами переходов из конкретного состояния.
/// </returns>
private Hashtable CreateTransitions( Automaton automaton )
{
    Hashtable hash = new Hashtable();
    foreach ( Transition t in automaton.Transitions )
    {
        if ( t.IsFromGroup )
        {
            foreach ( State st in automaton.Groups[ t.From ].States )
            {
                AddTransition( hash, st, t );
            }
        }
        else
        {
            State st = automaton.States[ t.From ];
```

```
                    AddTransition( hash, st, t );
                }
            }
            return hash;
        }

        /// <summary>
        /// Обход дерева состояний в глубину с целью выявления
        /// недостижимых состояний.
        /// </summary>
        /// <param name="automaton">Проверяемый автомат.</param>
        /// <param name="states">Непосещенные состояния.</param>
        /// <param name="state">
        /// Состояние, в которое осуществляется переход.
        /// </param>
        private void Walk( Automaton automaton, StateSet states, State state )
        {
            if ( !states.Contains( state ) )
            {
                return;
            }
            states.Remove( state );
            foreach ( Transition t in (IEnumerable)m_Transitions[ state ] )
            {
                Walk( automaton, states, automaton.States[ t.To ] );
            }
        }

        #endregion

    }
}
```

П1.38. StateGroup.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

using LoCoL.Collections;

namespace LoCoL
{
    /// <summary>
    /// Группа состояний автомата.
    /// </summary>
    [Serializable]
    public class StateGroup : NamedObject, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public StateGroup( int id, string name )
            : this( id, name, null )
```

```
{
}

/// <summary>
/// Конструктор.
/// </summary>
public StateGroup( int id, string name, StateSet states )
    : base( id, name )
{
    m_States =
        ( states == null ) ? ( new StateSet() ) : ( states.Clone() );
    m_States.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected StateGroup(
    SerializationInfo info,
    StreamingContext context )
    : base( info, context )
{
    m_States = (StateSet)info.GetValue( "States", typeof(StateSet) );
    m_States.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
    info.AddValue( "States", m_States );
}

#endregion

#region Private Members

/// <summary>
/// Состояния, которые входят в группу.
/// </summary>
private StateSet m_States;

#endregion

#region Public Properties

/// <summary>
/// Состояния, которые входят в группу.
/// </summary>
public StateSet States
{
    get
    {
```

```
            return m_States;
        }
    }

    #endregion

    #region IChangeable Implementation

    /// <summary>
    /// Событие, сообщающее, что набор состояний был изменен.
    /// </summary>
    public event ChangedEventHandler Changed;

    private void RaiseChangedEvent( object sender, ChangedEventArgs e )
    {
        if ( Changed != null )
        {
            Changed( this, e );
        }
    }

    #endregion
  }
}
```

П1.39. StateGroupSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор групп состояний.
    /// </summary>
    [Serializable]
    public class StateGroupSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public StateGroupSet()
        {
            m_Set = new NamedObjectSet();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        public StateGroupSet( StateGroupSet groups )
        {
```

```
    m_Set = groups.m_Set.Clone();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected StateGroupSet(
    SerializationInfo info,
    StreamingContext context )
{
    m_Set = (NamedObjectSet)info.GetValue(
        "Value",
        typeof(NamedObjectSet) );
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
public StateGroupSet Clone()
{
    return new StateGroupSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
```

```
/// Добавление группы состояния.
/// </summary>
public void Add( StateGroup group )
{
    m_Set.Add( group );
}

/// <summary>
/// Проверяет содержится ли группа состояний в коллекции.
/// </summary>
public bool Contains( StateGroup group )
{
    return m_Set.Contains( group );
}

/// <summary>
/// Удаляет группу состояний из коллекции.
/// Если группа состояний отсутствует, генерируется исключение.
/// </summary>
public void Remove( StateGroup group )
{
    m_Set.Remove( group );
}

#endregion

#region Public Properties

/// <summary>
/// Получение группы состояний по ее ID.
/// ID - свойство самой группы, а не ее позиция в коллекции.
/// </summary>
public StateGroup this[ int id ]
{
    get
    {
        return (StateGroup)m_Set[ id ];
    }
}

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}

#endregion

#region IEnumerable Implementation

/// <summary>
/// Enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
    return m_Set.GetEnumerator();
```

```
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор автоматов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.40. StateSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор состояний.
    /// </summary>
    [Serializable]
    public class StateSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public StateSet()
        {
            m_Set = new NamedObjectSet();
            m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }

        /// <summary>
        /// Конструктор копирования.
        /// </summary>
        /// <param name="states"></param>
        public StateSet( StateSet states )
        {
```

```
    m_Set = states.m_Set.Clone();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected StateSet( SerializationInfo info, StreamingContext context )
{
    m_Set = (NamedObjectSet)info.GetValue(
        "Value",
        typeof(NamedObjectSet) );
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создает копию.
/// </summary>
/// <returns></returns>
public StateSet Clone()
{
    return new StateSet( this );
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
/// Добавление состояния.
```

```
/// </summary>
public void Add( State state )
{
    m_Set.Add( state );
}

/// <summary>
/// Проверяет, содержится ли состояние в коллекции.
/// </summary>
public bool Contains( State state )
{
    return m_Set.Contains( state );
}

/// <summary>
/// Удаляет состояние из коллекции.
/// Если состояние отсутствует, генерируется исключение.
/// </summary>
public void Remove( State state )
{
    m_Set.Remove( state );
}

#endregion

#region Public Properties

/// <summary>
/// Получение состояния по его ID.
/// ID - свойство самого состояния, а не его позиция в коллекции.
/// </summary>
public State this[ int id ]
{
    get
    {
        return (State)m_Set[ id ];
    }
}

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}

#endregion

#region IEnumerable Implementation

/// <summary>
/// Enumerator.
/// </summary>
public IEnumerator GetEnumerator()
{
    return m_Set.GetEnumerator();
}
```

```
        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор состояний был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.41. TextWriteLogger.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.IO;

namespace LoCoL.Logs
{
    /// <summary>
    /// Класс ведущий протокол в простой текстовой форме.
    /// </summary>
    public class TextWriteLogger : ILogger
    {

        #region Public Constructor

        /// <summary>
        /// Конструктор.
        /// </summary>
        public TextWriteLogger( TextWriter writer )
        {
            m_Writer = writer;
            m_CallNesting = -1;
        }

        #endregion

        #region Private Members

        /// <summary>
        /// TextWriter, в который осуществляется протоколирование.
        /// </summary>
        private TextWriter m_Writer;
```

```
/// <summary>
/// Степень вложенности вызовов автоматов.
/// </summary>
private int m_CallNesting;

#endregion

#region Public Properties

/// <summary>
/// TextWriter, в который осуществляется протоколирование.
/// </summary>
public TextWriter Writer
{
    get
    {
        return m_Writer;
    }
}

#endregion

#region ILogger Implementation

/// <summary>
/// Протоколирование начала элементарного цикла.
/// </summary>
void ILogger.OnBeginCycle( AutomatonManager manager )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledBeginCycle( manager ) )
    {
        return;
    }
    Log( '[',
        "Начало цикла номер " + manager.CurrentCycle.ToString() );
}

/// <summary>
/// Протоколирование запуска автомата с некоторым уведомлением.
/// </summary>
void ILogger.OnBeginNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    m_CallNesting++;
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledBeginNotificationProcessing(
            automaton,
            notification ) )
    {
        return;
    }
    Log( '{',
        "Автомат " + automaton.ToString() +
        " приступил к обработке уведомления " +
        notification.ToString() +
        " в состоянии " + automaton.CurrentState.ToString() );
}

/// <summary>
```

```
/// Протоколирование опроса входной переменной.
/// </summary>
void ILogger.OnInputInquire(
    Automaton automaton,
    InputVariable variable,
    bool result )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledInputInquire( automaton, variable ) )
    {
        return;
    }
    Log( '<',
        "Автомат " + automaton.ToString() +
        " опросил входную переменную " + variable.ToString() +
        " и получил результат: " + ( result ? "истина" : "ложь" ) );
}

/// <summary>
/// Протоколирование сравнения переданного уведомления.
/// </summary>
void ILogger.OnNotificationChecking(
    Automaton automaton,
    Notification notification,
    bool result )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledNotificationChecking(
            automaton,
            notification ) )
    {
        return;
    }
    Log( 'N',
        "Автомат " + automaton.ToString() +
        " сравнил переданное уведомление с " + notification.ToString() +
        " и получил результат: " + ( result ? "истина" : "ложь" ) );
}

/// <summary>
/// Протоколирование сравнения состояния некоторого автомата в системе.
/// </summary>
void ILogger.OnStateChecking(
    Automaton automaton,
    Automaton checkedAutomaton,
    State state,
    bool result )
{
    if ( m_LogDetailer != null &&
        m_LogDetailer.IsEnabledStateChecking(
            automaton,
            checkedAutomaton,
            state ) )
    {
        return;
    }
    Log( 'S',
        "Автомат " + automaton.ToString() +
        " сравнил состояние автомата " + checkedAutomaton.ToString() +
        " с " + state.ToString() +
        " и получил результат: " + ( result ? "истина" : "ложь" ) );
```

```
}

/// <summary>
/// Протоколирование выполнения перехода.
/// </summary>
void ILogger.OnTransition( Automaton automaton, Transition transition )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledTransition( automaton, transition ) )
    {
        return;
    }
    Log( 'T',
        "Автомат " + automaton.ToString() +
        " совершил переход " + transition.ToString( automaton ) );
}

/// <summary>
/// Протоколирование вызова выходного воздействия.
/// </summary>
void ILogger.OnOutputInvocation(
    Automaton automaton,
    OutputAction action )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledOutputInvocation( automaton, action ) )
    {
        return;
    }
    Log( '>',
        "Автомат " + automaton.ToString() +
        " выполнил выходное воздействие " + action.ToString() );
}

/// <summary>
/// Протоколирование посылки сообщения.
/// </summary>
void ILogger.OnSendMessage(
    Automaton targetAutomaton,
    Notification notification )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledSendMessage(
            targetAutomaton,
            notification ) )
    {
        return;
    }
    Log( '!',
        "Автомату " + targetAutomaton.ToString() +
        " было послано сообщение " + notification.ToString() );
}

/// <summary>
/// Протоколирование завершения обработки уведомления автоматом.
/// </summary>
void ILogger.OnEndNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    if ( m_LogDetailer != null &&
```

```
            !m_LogDetailer.IsEnabledEndNotificationProcessing(
                automaton,
                notification ) )
        {
            m_CallNesting--;
            return;
        }
        Log( '}',
            "Автомат " + automaton.ToString() +
            " завершил обработку уведомления " + notification.ToString() +
            " в состоянии " + automaton.CurrentState.ToString() );
        m_CallNesting--;
    }

    /// <summary>
    /// Протоколирование окончания элементарного цикла.
    /// </summary>
    void ILogger.OnEndCycle( AutomatonManager manager )
    {
        if ( m_LogDetailer != null &&
            !m_LogDetailer.IsEnabledEndCycle( manager ) )
        {
            return;
        }
        Log( ']',
            "Окончание цикла номер " + manager.CurrentCycle );
    }

    private ILogDetailer m_LogDetailer;
    /// <summary>
    /// Класс, контролирующий детальность лога.
    /// </summary>
    public ILogDetailer LogDetailer
    {
        get
        {
            return m_LogDetailer;
        }
        set
        {
            m_LogDetailer = value;
        }
    }

    #endregion

    #region Private Methods

    /// <summary>
    /// Непосредственно протоколирование.
    /// </summary>
    /// <param name="typeOfMessage">Символ типа сообщения.</param>
    /// <param name="message">Сообщение.</param>
    private void Log( char typeOfMessage, string message )
    {
        m_Writer.Write( "[" + DateTime.Now.ToLongTimeString() + "] " );
        for ( int i = 0; i < m_CallNesting; i++ )
        {
            m_Writer.Write( '\t' );
        }
        m_Writer.WriteLine( typeOfMessage + " " + message );
```

```
            m_Writer.Flush();
        }

        #endregion

    }
}
```

П1.42. Transition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Reflection;
using System.Runtime.Serialization;

using LoCoL.Conditions;
using LoCoL.Collections;
using LoCoL.Exceptions;

namespace LoCoL
{
    /// <summary>
    /// Переход.
    /// </summary>
    [Serializable]
    public class Transition :
        NamedObject, IValidCheckable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
        public Transition(
            int id,
            string name,
            int from,
            bool isFromGroup,
            int to,
            int priority,
            Condition condition,
            AutomatonActionArray actions )
            : base( id, name )
        {
            m_From = from;
            m_IsFromGroup = isFromGroup;
            m_To = to;
            m_Priority = priority;
            m_Condition = condition;
            m_Actions =
                ( actions == null ) ?
                ( new AutomatonActionArray() ) :
                ( actions.Clone() );
            m_Actions.Changed += new ChangedEventHandler( RaiseChangedEvent );
        }
```

```
/// <summary>
/// Конструктор.
/// </summary>
public Transition(
    int id,
    string name,
    int from,
    int to,
    int priority,
    Condition condition,
    AutomatonActionArray actions )
    : this( id, name, from, false, to, priority, condition, actions )
{
}

/// <summary>
/// Конструктор.
/// </summary>
public Transition(
    int id,
    string name,
    int from,
    int to,
    Condition condition,
    AutomatonActionArray actions )
    : this( id, name, from, false, to, 0, condition, actions )
{
}

/// <summary>
/// Конструктор.
/// </summary>
public Transition(
    int id,
    string name,
    int from,
    int to,
    Condition condition )
    : this( id, name, from, false, to, 0, condition, null )
{
}

/// <summary>
/// Конструктор.
/// </summary>
public Transition( int id, string name, int from, int to )
    : this( id, name, from, false, to, 0, null, null )
{
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected Transition( SerializationInfo info, StreamingContext context )
    : base( info, context )
{
    m_From = (int)info.GetValue( "From", typeof(int) );
```

```
    m_IsFromGroup = (bool)info.GetValue( "IsFromGroup", typeof(bool) );
    m_To = (int)info.GetValue( "To", typeof(int) );
    m_Priority = (int)info.GetValue( "Priority", typeof(int) );
    m_Condition = (Condition)info.GetValue(
        "Condition",
        typeof(Condition) );
    m_Actions = (AutomatonActionArray)info.GetValue(
        "Actions",
        typeof(AutomatonActionArray) );
    m_Actions.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public override void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    base.GetObjectData( info, context );
    info.AddValue( "From", m_From );
    info.AddValue( "IsFromGroup", m_IsFromGroup );
    info.AddValue( "To", m_To );
    info.AddValue( "Priority", m_Priority );
    info.AddValue( "Condition", m_Condition );
    info.AddValue( "Actions", m_Actions );
}

#endregion

#region Private Members

/// <summary>
/// <para>
/// Номер состояния (или группы состояний), из которого (которой)
/// ведет переход.
/// </para>
/// <para>
/// Смотрите также <see cref="IsFromGroup"/>.
/// </para>
/// </summary>
private int m_From;

/// <summary>
/// Является ли переход из группы состояний (в противном случае,
/// переход из состояния).
/// </summary>
private bool m_IsFromGroup;

/// <summary>
/// Номер состояния, в который ведёт переход.
/// </summary>
private int m_To;

/// <summary>
/// Приоритет перехода. Максимальным считается приоритет 0.
/// Приоритет должен быть неотрицательным.
/// </summary>
private int m_Priority;

/// <summary>
```

```
/// Условие, привязанное к переходу.
/// Выполнение этого условия необходимо для осуществления перехода.
/// Если не установленно (т.е. равно null), то считается выполненным.
/// </summary>
private Condition m_Condition;

/// <summary>
/// Действия, выполняемые на переходе.
/// </summary>
private AutomatonActionArray m_Actions;

#endregion

#region Public Properties

/// <summary>
/// <para>
/// Номер состояния (или группы состояний), из которого (которой)
/// ведет переход.
/// </para>
/// <para>
/// Смотрите также <see cref="IsFromGroup"/>.
/// </para>
/// </summary>
public int From
{
    get
    {
        return m_From;
    }
}

/// <summary>
/// Является ли переход из группы состояний (в противном случае,
/// переход из состояния).
/// </summary>
public bool IsFromGroup
{
    get
    {
        return m_IsFromGroup;
    }
}

/// <summary>
/// Номер состояния, в который ведет переход.
/// </summary>
public int To
{
    get
    {
        return m_To;
    }
}

/// <summary>
/// Приоритет перехода. Максимальным считается приоритет 0.
/// Приоритет должен быть неотрицательным.
/// </summary>
/// <exception cref="TransitionException">
/// Приоритет перехода должен быть неотрицательным.
```

```
/// </exception>
public int Priority
{
    get
    {
        return m_Priority;
    }
    set
    {
        if ( value < 0 )
        {
            throw new TransitionException(
                "Приоритет перехода должен быть неотрицательным." );
        }
        m_Priority = value;
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
}

/// <summary>
/// Условие, привязанное к переходу.
/// Выполнение этого условия необходимо для осуществления перехода.
/// Если не установленно (т.е. равно null), то считается выполненным.
/// </summary>
public Condition Condition
{
    get
    {
        return m_Condition;
    }
    set
    {
        m_Condition = value;
        RaiseChangedEvent( this, ChangedEventArgs.Empty );
    }
}

/// <summary>
/// Действия, выполняемые на переходе.
/// </summary>
public AutomatonActionArray Actions
{
    get
    {
        return m_Actions;
    }
}

#endregion

#region Public Methods

/// <summary>
/// Строковое описание перехода с описание откуда и куда этот переход.
/// </summary>
/// <param name="automaton">
/// Автомат, которому принадлежит переход.
/// </param>
/// <returns>Описание.</returns>
public string ToString( Automaton automaton )
{
```

```
    string result = base.ToString();
    result += "[из ";
    result +=
        ( IsFromGroup ) ?
        ( "группы состояний " + automaton.Groups[ From ].ToString() ) :
        ( "состояния " + automaton.States[ From ].ToString() );
    result += " в состояние " + automaton.States[ To ].ToString() + "]";
    return result;
}

#endregion

#region IValidCheckable Implementation

/// <summary>
/// Проверить адекватность состояния перехода.
/// </summary>
/// <param name="args">Требуется <see cref="Automaton"/>.</param>
/// <exception cref="CheckValidException">
/// <list>
/// <item><description>
/// Метод требует единственный параметр типа <see cref="Automaton"/>.
/// </description></item>
/// <item><description>
/// Переход идет из несуществующей группы.
/// </description></item>
/// <item><description>
/// Переход идет из несуществующего состояния.
/// </description></item>
/// <item><description>
/// Переход ведет в несуществующее состояние.
/// </description></item>
/// </list>
/// </exception>
public void CheckValid( params object[] args )
{
    if ( args.Length != 1 || !( args[ 0 ] is Automaton ) )
    {
        throw new CheckValidException( "Метод " +
            this.GetType().Name + "." +
            MethodInfo.GetCurrentMethod().Name +
            " требует единственный параметр типа Automaton." );
    }
    Automaton a = (Automaton)args[ 0 ];
    if ( IsFromGroup && a.Groups[ From ] == null )
    {
        throw new CheckValidException( "Переход " + ToString() +
            " идет из несуществующей группы с ID==" +
            From + " автомата " + a.ToString() + "." );
    }
    if ( !IsFromGroup && a.States[ From ] == null )
    {
        throw new CheckValidException( "Переход " + ToString() +
            " идет из несуществующего состояния с ID==" +
            From + " автомата " + a.ToString() + "." );
    }
    if ( a.States[ To ] == null )
    {
        throw new CheckValidException( "Переход " + ToString() +
            " ведет в несуществующее состояние с ID==" +
            From + " автомата " + a.ToString() + "." );
```

```
            }
            if ( Condition != null )
            {
                Condition.CheckValid( args[ 0 ] );
            }
            foreach ( IAutomatonAction aa in Actions )
            {
                aa.CheckValid( a );
            }
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор действий автомата был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        /// <summary>
        /// Посылка сообщения <see cref="Changed"/>.
        /// </summary>
        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.43. TransitionSet.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Collections;
using System.Runtime.Serialization;

namespace LoCoL.Collections
{
    /// <summary>
    /// Набор переходов.
    /// </summary>
    [Serializable]
    public class TransitionSet : IEnumerable, IChangeable, ISerializable
    {

        #region Public Constructors

        /// <summary>
        /// Конструктор.
        /// </summary>
```

```
public TransitionSet()
{
    m_Set = new NamedObjectSet();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Конструктор копирования.
/// </summary>
public TransitionSet( TransitionSet transitions )
{
    m_Set = transitions.m_Set.Clone();
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

#endregion

#region ISerializable Implementation

/// <summary>
/// Конструктор десериализации.
/// </summary>
protected TransitionSet(
    SerializationInfo info,
    StreamingContext context )
{
    m_Set = (NamedObjectSet)info.GetValue(
        "Value",
        typeof(NamedObjectSet) );
    m_Set.Changed += new ChangedEventHandler( RaiseChangedEvent );
}

/// <summary>
/// Контроль над сериализацией объекта.
/// </summary>
public virtual void GetObjectData(
    SerializationInfo info,
    StreamingContext context )
{
    info.AddValue( "Value", m_Set );
}

#endregion

#region Private Members

/// <summary>
/// Внутренняя коллекция именованных объектов.
/// </summary>
private NamedObjectSet m_Set;

#endregion

#region Public Methods

/// <summary>
/// Создание копии.
/// </summary>
public TransitionSet Clone()
{
    return new TransitionSet( this );
```

```
}

/// <summary>
/// Получение перехода по его ID.
/// ID - свойство самого перехода, а не его позиция в коллекции.
/// </summary>
public Transition this[ int id ]
{
    get
    {
        return (Transition)m_Set[ id ];
    }
}

/// <summary>
/// Очистить коллекцию.
/// </summary>
public void Clear()
{
    m_Set.Clear();
}

/// <summary>
/// Добавление перехода.
/// </summary>
public void Add( Transition transition )
{
    m_Set.Add( transition );
}

/// <summary>
/// Проверяет, содержится ли переход в коллекции.
/// </summary>
public bool Contains( Transition transition )
{
    return m_Set.Contains( transition );
}

/// <summary>
/// Удаляет переход из коллекции.
/// Если переход отсутствует, генерируется исключение.
/// </summary>
public void Remove( Transition transition )
{
    m_Set.Remove( transition );
}

#endregion

#region Public Properties

/// <summary>
/// Количество состояний в коллекции.
/// </summary>
public int Count
{
    get
    {
        return m_Set.Count;
    }
}
```

```
        #endregion

        #region IEnumerable Implementation

        /// <summary>
        /// Enumerator.
        /// </summary>
        public IEnumerator GetEnumerator()
        {
            return m_Set.GetEnumerator();
        }

        #endregion

        #region IChangeable Implementation

        /// <summary>
        /// Событие, сообщающее, что набор переходов был изменен.
        /// </summary>
        public event ChangedEventHandler Changed;

        /// <summary>
        /// Посылка сообщения <see cref="Changed"/>.
        /// </summary>
        private void RaiseChangedEvent( object sender, ChangedEventArgs e )
        {
            if ( Changed != null )
            {
                Changed( this, e );
            }
        }

        #endregion

    }
}
```

П1.44. XmlLogger.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Xml;

using LoCoL.Exceptions;

namespace LoCoL.Logs
{

    /// <summary>
    /// Класс ведущий протокол в форме XML-файла.
    /// </summary>
    public class XmlLogger : ILogger, IDisposable
    {

        #region Public Constructor
```

```
/// <summary>
/// Конструктор.
/// Сам регистрирует себя у менеджера автоматов.
/// </summary>
/// <param name="writer">
/// <see cref="XmlWriter"/>, в который ведется протокол.
/// </param>
/// <param name="manager">
/// <see cref="AutomatonManager"/>, к которому привязан
/// <see cref="XmlLogger"/>.
/// </param>
/// <exception cref="LoggerException">
/// <list>
/// <item><description>
/// <c>writer</c> не может быть <c>null</c>.
/// </description></item>
/// <item><description>
/// <c>manager</c> не может быть <c>null</c>.
/// </description></item>
/// </list>
/// </exception>
public XmlLogger( XmlWriter writer, AutomatonManager manager )
{
    if ( writer == null )
    {
        throw new LoggerException( "XmlWriter не может быть null." );
    }
    if ( manager == null )
    {
        throw new LoggerException(
            "AutomatonManager не может быть null." );
    }
    m_Writer = writer;
    m_Writer.WriteStartDocument();
    m_Manager = manager;
    m_Manager.RegisterLogger( this );
    m_Writer.WriteStartElement( "AutomatonLog" );
    m_Writer.WriteAttributeString(
        "Date",
        DateTime.Now.ToShortDateString() );
    if ( m_Manager.Title != null )
    {
        m_Writer.WriteAttributeString( "Title", m_Manager.Title );
    }
    if ( m_Manager.Version != null )
    {
        m_Writer.WriteAttributeString(
            "Version",
            m_Manager.Version.ToString() );
    }
    if ( m_Manager.Description != null )
    {
        m_Writer.WriteAttributeString(
            "Description",
            m_Manager.Description );
    }
    if ( m_Manager.Copyright != null )
    {
        m_Writer.WriteAttributeString(
            "Copyright",
            m_Manager.Copyright );
```

```
    }
}

#endregion

#region Destructor

/// <summary>
/// Деструктор.
/// </summary>
~XmlLogger()
{
    Dispose();
}

#endregion

#region Private Members

/// <summary>
/// <see cref="XmlWriter"/>, в который осуществляется протоколирование.
/// </summary>
private XmlWriter m_Writer;

/// <summary>
/// Объект, контролирующий детальность лога.
/// </summary>
private ILogDetailer m_LogDetailer;

/// <summary>
/// Менеджер автоматов, для которого создан <see cref="XmlLogger"/>.
/// </summary>
private AutomatonManager m_Manager;

#endregion

#region Public Properties

/// <summary>
/// <see cref="XmlWriter"/>, в который осуществляется протоколирование.
/// </summary>
public XmlWriter Writer
{
    get
    {
        return m_Writer;
    }
}

/// <summary>
/// Менеджер автоматов, для которого создан <see cref="XmlLogger"/>.
/// </summary>
public AutomatonManager Manager
{
    get
    {
        return m_Manager;
    }
}

#endregion
```

```
#region ILogger Implementation

/// <summary>
/// Протоколирование начала элементарного цикла.
/// </summary>
void ILogger.OnBeginCycle( AutomatonManager manager )
{
    m_Writer.WriteStartElement( "Cycle" );
    m_Writer.WriteAttributeString(
        "Tick",
        manager.CurrentCycle.ToString() );
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledBeginCycle( manager ) )
    {
        return;
    }
}

/// <summary>
/// Протоколирование запуска автомата с некоторым уведомлением.
/// </summary>
void ILogger.OnBeginNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    m_Writer.WriteStartElement( "Automaton" );
    m_Writer.WriteAttributeString( "Name", automaton.ToString() );
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledBeginNotificationProcessing(
            automaton,
            notification ) )
    {
        return;
    }
    m_Writer.WriteStartElement( "BeginNotificationProcessing" );
    m_Writer.WriteAttributeString(
        "Time",
        DateTime.Now.ToLongTimeString() );
    m_Writer.WriteAttributeString(
        "Notification",
        notification.ToString() );
    m_Writer.WriteAttributeString(
        "State",
        automaton.CurrentState.ToString() );
    m_Writer.WriteEndElement();
}

/// <summary>
/// Протоколирование опроса входной переменной.
/// </summary>
void ILogger.OnInputInquire(
    Automaton automaton,
    InputVariable variable,
    bool result )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledInputInquire( automaton, variable ) )
    {
        return;
    }
```

```
    m_Writer.WriteStartElement( "InputInquire" );
    m_Writer.WriteAttributeString(
        "Time",
        DateTime.Now.ToLongTimeString() );
    m_Writer.WriteAttributeString(
        "InputVariable",
        variable.ToString() );
    m_Writer.WriteAttributeString(
        "Result",
        result ? "истина" : "ложь" );
    m_Writer.WriteEndElement();
}

/// <summary>
/// Протоколирование сравнения переданного уведомления.
/// </summary>
void ILogger.OnNotificationChecking(
    Automaton automaton,
    Notification notification,
    bool result )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledNotificationChecking(
            automaton,
        notification ) )
    {
        return;
    }
    m_Writer.WriteStartElement( "NotificationChecking" );
    m_Writer.WriteAttributeString(
        "Time",
        DateTime.Now.ToLongTimeString() );
    m_Writer.WriteAttributeString(
        "Notification",
        notification.ToString() );
    m_Writer.WriteAttributeString(
        "Result", result ? "истина" : "ложь" );
    m_Writer.WriteEndElement();
}

/// <summary>
/// Протоколирование сравнения состояния некоторого автомата в системе.
/// </summary>
void ILogger.OnStateChecking(
    Automaton automaton,
    Automaton checkedAutomaton,
    State state,
    bool result )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledStateChecking(
            automaton,
            checkedAutomaton,
            state ) )
    {
        return;
    }
    m_Writer.WriteStartElement( "StateChecking" );
    m_Writer.WriteAttributeString(
        "Time",
        DateTime.Now.ToLongTimeString() );
```

```
            m_Writer.WriteAttributeString(
                "CheckedAutomaton",
                checkedAutomaton.ToString() );
            m_Writer.WriteAttributeString(
                "State",
                state.ToString() );
            m_Writer.WriteAttributeString(
                "Result",
                result ? "истина" : "ложь" );
            m_Writer.WriteEndElement();
        }

        /// <summary>
        /// Протоколирование выполнения перехода.
        /// </summary>
        void ILogger.OnTransition( Automaton automaton, Transition transition )
        {
            if ( m_LogDetailer != null &&
                !m_LogDetailer.IsEnabledTransition( automaton, transition ) )
            {
                return;
            }
            m_Writer.WriteStartElement( "Transition" );
            m_Writer.WriteAttributeString(
                "Time",
                DateTime.Now.ToLongTimeString() );
            m_Writer.WriteAttributeString(
                "Transition",
                transition.ToString( automaton ) );
            m_Writer.WriteEndElement();
        }

        /// <summary>
        /// Протоколирование посылки сообщения.
        /// </summary>
        void ILogger.OnSendMessage( Automaton targetAutomaton, Notification
notification )
        {
            if ( m_LogDetailer != null &&
                !m_LogDetailer.IsEnabledSendMessage(
                    targetAutomaton,
                    notification ) )
            {
                return;
            }
            m_Writer.WriteStartElement( "SendMessage" );
            m_Writer.WriteAttributeString(
                "Time",
                DateTime.Now.ToLongTimeString() );
            m_Writer.WriteAttributeString(
                "TargetAutomaton",
                targetAutomaton.ToString() );
            m_Writer.WriteAttributeString(
                "Message",
                notification.ToString() );
            m_Writer.WriteEndElement();
        }

        /// <summary>
        /// Протоколирование вызова выходного воздействия.
        /// </summary>
```

```
void ILogger.OnOutputInvocation(
    Automaton automaton,
    OutputAction action )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledOutputInvocation( automaton, action ) )
    {
        return;
    }
    m_Writer.WriteStartElement( "OutputInvocation" );
    m_Writer.WriteAttributeString(
        "Time",
        DateTime.Now.ToLongTimeString() );
    m_Writer.WriteAttributeString(
        "OutputAction",
        action.ToString() );
    m_Writer.WriteEndElement();
}

/// <summary>
/// Протоколирование завершения обработки уведомления автоматом.
/// </summary>
void ILogger.OnEndNotificationProcessing(
    Automaton automaton,
    Notification notification )
{
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledEndNotificationProcessing(
            automaton,
            notification ) )
    {
    }
    else
    {
        m_Writer.WriteStartElement( "EndNotificationProcessing" );
        m_Writer.WriteAttributeString(
            "Time",
            DateTime.Now.ToLongTimeString() );
        m_Writer.WriteAttributeString(
            "Notification",
            notification.ToString() );
        m_Writer.WriteAttributeString(
            "State",
            automaton.CurrentState.ToString() );
        m_Writer.WriteEndElement();
    }
    // Закрывает тэг Automaton.
    m_Writer.WriteEndElement();
}

/// <summary>
/// Протоколирование окончания элементарного цикла.
/// </summary>
void ILogger.OnEndCycle( AutomatonManager manager )
{
    // Закрывает тэг Cycle.
    m_Writer.WriteEndElement();
    if ( m_LogDetailer != null &&
        !m_LogDetailer.IsEnabledEndCycle( manager ) )
    {
        return;
```

```
            }
        }

        /// <summary>
        /// Объект, контролирующий детальность лога.
        /// </summary>
        public ILogDetailer LogDetailer
        {
            get
            {
                return m_LogDetailer;
            }
            set
            {
                m_LogDetailer = value;
            }
        }

        #endregion

        #region IDisposable Implementation

        /// <summary>
        /// Флаг запуска метода <see cref="Dispose"/>.
        /// </summary>
        private bool m_Disposed = false;
        /// <summary>
        /// Освобождает занятые ресурсы. Не забывайте вызывать этот метод.
        /// Возложение вызова <see cref="Dispose"/> на деструктор
        /// не гарантирует правильности работы.
        /// </summary>
        public void Dispose()
        {
            if ( !m_Disposed )
            {
                // Закрывает тэг AutomatonLog.
                m_Writer.WriteEndElement();
                m_Writer.WriteEndDocument();
                m_Writer.Close();
                m_Disposed = true;
            }
        }

        #endregion
    }
}
```

П1.45. XorCondition.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Runtime.Serialization;

namespace LoCoL.Conditions
{

    /// <summary>
```

```
/// Логическое исключающее ИЛИ.
/// </summary>
[Serializable]
public class XorCondition : Condition, ISerializable
{

    #region Public Constructor

    /// <summary>
    /// Конструктор.
    /// </summary>
    /// <param name="c1">Первый операнд.</param>
    /// <param name="c2">Второй операнд.</param>
    public XorCondition( Condition c1, Condition c2 )
        : base( c1, c2 )
    {
    }

    #endregion

    #region ISerializable Implementation

    /// <summary>
    /// Конструктор десериализации.
    /// </summary>
    protected XorCondition(
        SerializationInfo info,
        StreamingContext context )
        : base( info, context )
    {
    }

    /// <summary>
    /// Контроль над сериализацией объекта.
    /// </summary>
    public override void GetObjectData(
        SerializationInfo info,
        StreamingContext context )
    {
        base.GetObjectData( info, context );
    }

    #endregion

    #region Condition Overrided Methods

    /// <summary>
    /// Вычисление выражения "логическое исключающее ИЛИ"
    /// в контексте определенного автомата.
    /// </summary>
    /// <param name="automaton">Автомат, в контексте которого выполняется
    /// вычисление выражения.</param>
    /// <returns>Результат вычисления.</returns>
    public override bool Evaluate( Automaton automaton )
    {
        return ((Condition)this[ 0 ]).Evaluate( automaton ) ^
            ((Condition)this[ 1 ]).Evaluate( automaton );
    }

    #endregion
```

```
    }
}
```

ПРИЛОЖЕНИЕ 2. ПРОТОКОЛИРОВАНИЕ, XML И XSLT. ПРИМЕР ИСПОЛЬЗОВАНИЯ

П2.1. Листинг файла XSL Transformation (Default.xslt)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:template match="/AutomatonLog">
        <html>
            <head>
                <title>Протокол работы системы автоматов</title>
                <link rel="stylesheet" type="text/css" href="Default.css" />
                <script language="JavaScript" src="Default.js" />
            </head>
            <body>
                <div class="head">
                    <h1>Протокол работы системы автоматов</h1>
                    Название системы: <b><xsl:value-of select="@Title" /></b><br />
                    Версия системы: <b><xsl:value-of select="@Version" /></b><br />
                    Описание системы: <i><xsl:value-of select="@Description" Ð
/></i><br />
                    Авторское право: <b><xsl:value-of select="@Copyright" /></b>Ð
<br />
                    Дата создания протокола: <b><xsl:value-of select="@Date" /></b>
                </div>
                <xsl:for-each select="Cycle">
                    <xsl:apply-templates select="." />
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="Cycle">
        <script language="JavaScript">BeginDivClass(0);</script>
        <i>[Начало цикла номер <b><xsl:value-of select="@Tick" /></b>]</i>
        <xsl:for-each select="Automaton">
            <xsl:apply-templates select="." />
        </xsl:for-each>
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="Automaton">
        <script language="JavaScript">deepNested++;</script>
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">BeginDivClass(1);</script>
        <script language="JavaScript">Indent();</script>
        <i>[Работа автомата <b><xsl:value-of select="@Name" /></b>]</i>
        <xsl:for-each select="BeginNotificationProcessing|InputInquire|Ð
NotificationChecking|StateChecking|Transition|SendMessage|Automaton|Ð
OutputInvocation|EndNotificationProcessing">
            <xsl:apply-templates select="." />
        </xsl:for-each>
        <script language="JavaScript">EndDivClass(1);</script>
```

```
        <script language="JavaScript">EndDivClass(0);</script>
        <script language="JavaScript">deepNested--;</script>
    </xsl:template>

    <xsl:template match="BeginNotificationProcessing">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат приступил к обработке уведомления <b><xsl:value-of select=Ð
"@Notification"/></b> в состоянии <b><xsl:value-of select="@State"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="InputInquire">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат опросил входную переменную <b><xsl:value-of select=Ð
"@InputVariable"/></b> и получил результат: <b><xsl:value-of select=Ð
"@Result"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="NotificationChecking">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат сравнил переданное уведомление с <b><xsl:value-of select=Ð
"@Notification"/></b> и получил результат: <b><xsl:value-of select="@Result"/>Ð
</b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="StateChecking">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат сравнил состояние автомата <b><xsl:value-of select="Ð
@CheckedAutomaton"/></b> с <b><xsl:value-of select="@State"/></b> и получил Ð
результат: <b><xsl:value-of select="@Result"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="Transition">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат совершил переход <b><xsl:value-of select="@Transition"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="OutputInvocation">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат выполнил выходное воздействие <b><xsl:value-of select=Ð
"@OutputAction"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="SendMessage">
```

```
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомату <b><xsl:value-of select="@TargetAutomaton"/></b> было послано Ð
сообщение <b><xsl:value-of select="@Message"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="EndNotificationProcessing">
        <script language="JavaScript">BeginDivClass(0);</script>
        <script language="JavaScript">Indent();</script>
        <xsl:apply-templates select="@Time" />
        Автомат завершил обработку уведомления <b><xsl:value-of select=Ð
"@Notification"/></b> в состоянии <b><xsl:value-of select="@State"/></b>.
        <script language="JavaScript">EndDivClass(0);</script>
    </xsl:template>

    <xsl:template match="@Time">
        [<xsl:value-of select="." />]
    </xsl:template>
</xsl:stylesheet>
```

П2.2. Листинг файла JavaScript (Default.js)

```
var deepNested = -1;
var coloring0 = 0;
var coloring1 = 0;
function Indent()
{
    for ( var i = 0; i != deepNested; i++ )
    {
        document.write( '<span class="indent"></span>' );
    }
}
function BeginDivClass( flag )
{
    if ( flag == 0 )
    {
        coloring0 = 1 - coloring0;
    }
    if ( flag == 1 )
    {
        coloring1 = 1 - coloring1;
    }
    document.write( "<div class='log" + coloring1 + coloring0 + "'>" );
}

function EndDivClass( flag )
{
    if ( flag == 1 )
    {
        coloring1 = 1 - coloring1;
    }
    document.write( "</div>" );
}
```

П2.3. Листинг файла Cascading Style Sheet (Default.css)

```
body
{
    font-family: Courier New;
    font-size: 15px;
}
span.indent
{
    width: 50px;
}
h1
{
    font-family: Verdana;
    font-size: 20px;
}
div.head
{
    background-color: #DDDDDD;
}
div.log00
{
    background-color: #FFBFBF;
}
div.log01
{
    background-color: #D98D8D;
}
div.log10
{
    background-color: #BFC5FF;
}
div.log11
{
    background-color: #8D93D9;
}
```

П2.4. Листинг файла XML-протокола (Log.xml)

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet type="text/xsl" href="Default.xslt" ?>
<AutomatonLog Date="11.05.2004" Title="Some Title" Version="0.9" Description="Some
Description" Copyright="Some Copyright">
    <Cycle Tick="0">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:29"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
            <NotificationChecking Time="21:31:30" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="1">
        <Automaton Name="A0(Нулевой автомат)">
            <BeginNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
```

```
            <InputInquire Time="21:31:30" InputVariable="x1(Входная переменная 1)"
Result="истина" />
            <Transition Time="21:31:30" Transition="0(Noname)[из состояния
0(Noname) в состояние 1(Noname)]" />
            <SendMessage Time="21:31:30" TargetAutomaton="A0(Нулевой автомат)"
Message="m1(Уведомление1)" />
            <EndNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="2">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
            <NotificationChecking Time="21:31:30" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="3">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
            <NotificationChecking Time="21:31:30" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:30"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="4">
        <Automaton Name="A0(Нулевой автомат)">
            <BeginNotificationProcessing Time="21:31:31"
Notification="m1(Уведомление1)" State="1(Noname)" />
            <InputInquire Time="21:31:31" InputVariable="x1(Входная переменная 1)"
Result="истина" />
            <Transition Time="21:31:31" Transition="1(Noname)[из состояния
1(Noname) в состояние 2(Noname)]" />
            <Automaton Name="A1(Первый автомат)">
                <BeginNotificationProcessing Time="21:31:31"
Notification="e1(Уведомление1)" State="0(Noname)" />
                <NotificationChecking Time="21:31:31"
Notification="n1(Уведомление1)" Result="истина" />
                <Transition Time="21:31:31" Transition="0(Noname)[из состояния
0(Noname) в состояние 1(Noname)]" />
                <EndNotificationProcessing Time="21:31:31"
Notification="e1(Уведомление1)" State="1(Noname)" />
            </Automaton>
            <EndNotificationProcessing Time="21:31:31"
Notification="m1(Уведомление1)" State="2(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="5">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:31"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
            <StateChecking Time="21:31:31" CheckedAutomaton="A0(Нулевой автомат)"
State="1(Noname)" Result="ложь" />
            <NotificationChecking Time="21:31:31" Notification="n1(Уведомление1)"
Result="ложь" />
```

```
            <EndNotificationProcessing Time="21:31:31"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="6">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:31"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
            <StateChecking Time="21:31:31" CheckedAutomaton="A0(Нулевой автомат)"
State="1(Noname)" Result="ложь" />
            <NotificationChecking Time="21:31:31" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:31"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="7">
        <Automaton Name="A0(Нулевой автомат)">
            <BeginNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="2(Noname)" />
            <InputInquire Time="21:31:32" InputVariable="x1(Входная переменная 1)"
Result="истина" />
            <Transition Time="21:31:32" Transition="3(Noname)[из группы состояний
0(Noname) в состояние 0(Noname)]" />
            <EndNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="8">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
            <StateChecking Time="21:31:32" CheckedAutomaton="A0(Нулевой автомат)"
State="1(Noname)" Result="ложь" />
            <NotificationChecking Time="21:31:32" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="9">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
            <StateChecking Time="21:31:32" CheckedAutomaton="A0(Нулевой автомат)"
State="1(Noname)" Result="ложь" />
            <NotificationChecking Time="21:31:32" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:32"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="10">
        <Automaton Name="A0(Нулевой автомат)">
            <BeginNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
            <InputInquire Time="21:31:33" InputVariable="x1(Входная переменная 1)"
Result="истина" />
            <Transition Time="21:31:33" Transition="0(Noname)[из состояния
0(Noname) в состояние 1(Noname)]" />
```

```
            <SendMessage Time="21:31:33" TargetAutomaton="A0(Нулевой автомат)"
Message="m1(Уведомление1)" />
            <EndNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="11">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="1(Noname)" />
            <StateChecking Time="21:31:33" CheckedAutomaton="A0(Нулевой автомат)"
State="1(Noname)" Result="истина" />
            <Transition Time="21:31:33" Transition="1(Noname)[из состояния
1(Noname) в состояние 0(Noname)]" />
            <OutputInvocation Time="21:31:33" OutputAction="11(Вых_в_ие_11)" />
            <EndNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="12">
        <Automaton Name="A1(Первый автомат)">
            <BeginNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
            <NotificationChecking Time="21:31:33" Notification="n1(Уведомление1)"
Result="ложь" />
            <EndNotificationProcessing Time="21:31:33"
Notification="m0(&lt;Reserved&gt;)" State="0(Noname)" />
        </Automaton>
    </Cycle>
    <Cycle Tick="13">
        <Automaton Name="A0(Нулевой автомат)">
            <BeginNotificationProcessing Time="21:31:34"
Notification="m1(Уведомление1)" State="1(Noname)" />
            <InputInquire Time="21:31:34" InputVariable="x1(Входная переменная 1)"
Result="истина" />
            <Transition Time="21:31:34" Transition="1(Noname)[из состояния
1(Noname) в состояние 2(Noname)]" />
            <Automaton Name="A1(Первый автомат)">
                <BeginNotificationProcessing Time="21:31:34"
Notification="e1(Уведомление1)" State="0(Noname)" />
                <NotificationChecking Time="21:31:34"
Notification="n1(Уведомление1)" Result="истина" />
                <Transition Time="21:31:34" Transition="0(Noname)[из состояния
0(Noname) в состояние 1(Noname)]" />
                <EndNotificationProcessing Time="21:31:34"
Notification="e1(Уведомление1)" State="1(Noname)" />
            </Automaton>
            <EndNotificationProcessing Time="21:31:34"
Notification="m1(Уведомление1)" State="2(Noname)" />
        </Automaton>
    </Cycle>
</AutomatonLog>
```

П2.5. Результат применения XSLT к файлу протокола

Протокол работы системы автоматов

Название системы: **Some Title**
Версия системы: **0.9**
Описание системы: *Some Description*
Авторское право: **Some Copyright**
Дата создания протокола: **11.05.2004**

[Начало цикла номер 0]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:29] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:30] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:30] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[Начало цикла номер 1]
*[Работа автомата **A0(Нулевой автомат)**]*
[21:31:30] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:30] Автомат опросил входную переменную **x1(Входная переменная 1)** и получил результат: **истина**.
[21:31:30] Автомат совершил переход **0(Noname)[из состояния 0(Noname) в состояние 1(Noname)]**.
[21:31:30] Автомату **A0(Нулевой автомат)** было послано сообщение **m1(Уведомление1)**.
[21:31:30] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 2]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:30] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:30] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:30] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[Начало цикла номер 3]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:30] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:30] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:30] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[Начало цикла номер 4]
*[Работа автомата **A0(Нулевой автомат)**]*
[21:31:31] Автомат приступил к обработке уведомления **m1(Уведомление1)** в состоянии **1(Noname)**.
[21:31:31] Автомат опросил входную переменную **x1(Входная переменная 1)** и получил результат: **истина**.
[21:31:31] Автомат совершил переход **1(Noname)[из состояния 1(Noname) в состояние 2(Noname)]**.
    *[Работа автомата **A1(Первый автомат)**]*
    [21:31:31] Автомат приступил к обработке уведомления **e1(Уведомление1)** в состоянии **0(Noname)**.
    [21:31:31] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **истина**.
    [21:31:31] Автомат совершил переход **0(Noname)[из состояния 0(Noname) в состояние 1(Noname)]**.
    [21:31:31] Автомат завершил обработку уведомления **e1(Уведомление1)** в состоянии **1(Noname)**.
[21:31:31] Автомат завершил обработку уведомления **m1(Уведомление1)** в состоянии **2(Noname)**.
[Начало цикла номер 5]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:31] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[21:31:31] Автомат сравнил состояние автомата **A0(Нулевой автомат)** с **1(Noname)** и получил результат: **ложь**.
[21:31:31] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:31] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 6]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:31] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[21:31:31] Автомат сравнил состояние автомата **A0(Нулевой автомат)** с **1(Noname)** и получил результат: **ложь**.
[21:31:31] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:31] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 7]
*[Работа автомата **A0(Нулевой автомат)**]*
[21:31:32] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **2(Noname)**.
[21:31:32] Автомат опросил входную переменную **x1(Входная переменная 1)** и получил результат: **истина**.
[21:31:32] Автомат совершил переход **3(Noname)[из группы состояний 0(Noname) в состояние 0(Noname)]**.
[21:31:32] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.

[Начало цикла номер 8]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:32] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[21:31:32] Автомат сравнил состояние автомата **A0(Нулевой автомат)** с **1(Noname)** и получил результат: **ложь**.
[21:31:32] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:32] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 9]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:32] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[21:31:32] Автомат сравнил состояние автомата **A0(Нулевой автомат)** с **1(Noname)** и получил результат: **ложь**.
[21:31:32] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:32] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 10]
*[Работа автомата **A0(Нулевой автомат)**]*
[21:31:33] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:33] Автомат опросил входную переменную **x1(Входная переменная 1)** и получил результат: **истина**.
[21:31:33] Автомат совершил переход **0(Noname)[из состояния 0(Noname) в состояние 1(Noname)]**.
[21:31:33] Автомату **A0(Нулевой автомат)** было послано сообщение **m1(Уведомление1)**.
[21:31:33] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[Начало цикла номер 11]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:33] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **1(Noname)**.
[21:31:33] Автомат сравнил состояние автомата **A0(Нулевой автомат)** с **1(Noname)** и получил результат: **истина**.
[21:31:33] Автомат совершил переход **1(Noname)[из состояния 1(Noname) в состояние 0(Noname)]**.
[21:31:33] Автомат выполнил выходное воздействие **11(Вых_в_ие_11)**.
[21:31:33] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[Начало цикла номер 12]
*[Работа автомата **A1(Первый автомат)**]*
[21:31:33] Автомат приступил к обработке уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[21:31:33] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **ложь**.
[21:31:33] Автомат завершил обработку уведомления **m0(<Reserved>)** в состоянии **0(Noname)**.
[Начало цикла номер 13]
*[Работа автомата **A0(Нулевой автомат)**]*
[21:31:34] Автомат приступил к обработке уведомления **m1(Уведомление1)** в состоянии **1(Noname)**.
[21:31:34] Автомат опросил входную переменную **x1(Входная переменная 1)** и получил результат: **истина**.
[21:31:34] Автомат совершил переход **1(Noname)[из состояния 1(Noname) в состояние 2(Noname)]**.
    *[Работа автомата **A1(Первый автомат)**]*
    [21:31:34] Автомат приступил к обработке уведомления **e1(Уведомление1)** в состоянии **0(Noname)**.
    [21:31:34] Автомат сравнил переданное уведомление с **n1(Уведомление1)** и получил результат: **истина**.
    [21:31:34] Автомат совершил переход **0(Noname)[из состояния 0(Noname) в состояние 1(Noname)]**.
    [21:31:34] Автомат завершил обработку уведомления **e1(Уведомление1)** в состоянии **1(Noname)**.
[21:31:34] Автомат завершил обработку уведомления **m1(Уведомление1)** в состоянии **2(Noname)**.

ПРИЛОЖЕНИЕ 3. ИСХОДНЫЕ КОДЫ ПРИМЕРА ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ

П3.1. LoCoL_Demo_Create

П3.1.1. Class1.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

using LoCoL;
using LoCoL.Collections;
using LoCoL.Conditions;

namespace LoCoL_Demo_Create
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            AutomatonManager am = new AutomatonManager();

            InputVariable x0 =
                new InputVariable( 0, "Текущий символ - нулевой" );
            InputVariable x1 =
                new InputVariable( 1, "Текущий символ - цифра" );
            InputVariable x2 =
                new InputVariable( 2, "Текущий символ - пробел" );
            OutputAction z2 =
                new OutputAction( 2, "Перейти к следующему символу" );
            OutputAction z3 =
                new OutputAction( 3, "Вывести текущий символ" );
            OutputAction z4 =
                new OutputAction( 4, "Добавить символ в буфер" );
            OutputAction z5 =
                new OutputAction( 5, "Вывести буфер в обратном порядке" );
            OutputAction z6 =
                new OutputAction( 6, "Вывести буфер в прямом порядке" );
            OutputAction z7 =
                new OutputAction( 7, "Увеличить счетчик слов" );
            OutputAction z9 =
                new OutputAction( 9, "Процесс завершен" );

            Automaton a1 = new Automaton( 1, "Переворот чисел" );
            a1.InputVariables.Add( x0 );
            a1.InputVariables.Add( x1 );
            a1.InputVariables.Add( x2 );
            a1.OutputActions.Add( z3 );
```

```
a1.OutputActions.Add( z4 );
a1.OutputActions.Add( z5 );
a1.OutputActions.Add( z6 );
a1.States.Add( new State( 0, "Начальное состояние" ) );
a1.States.Add( new State( 1, "Ожидание" ) );
a1.States.Add( new State( 2, "Пробелы" ) );
a1.States.Add( new State( 3, "Число" ) );
a1.Transitions.Add( new Transition( 0, "0->0", 0, 0 ) );
a1.Transitions[ 0 ].Condition = x0;
a1.Transitions.Add( new Transition( 1, "0->1", 0, 1 ) );
a1.Transitions[ 1 ].Condition = !x0 & !x1 & !x2;
a1.Transitions[ 1 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 2, "0->2", 0, 2 ) );
a1.Transitions[ 2 ].Condition = x2;
a1.Transitions[ 2 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 3, "0->3", 0, 3 ) );
a1.Transitions[ 3 ].Condition = x1;
a1.Transitions[ 3 ].Actions.Add( z4 );
a1.Transitions.Add( new Transition( 4, "1->0", 1, 0 ) );
a1.Transitions[ 4 ].Condition = x0;
a1.Transitions.Add( new Transition( 5, "1->1", 1, 1 ) );
a1.Transitions[ 5 ].Condition = !x0 & !x2;
a1.Transitions[ 5 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 6, "1->2", 1, 2 ) );
a1.Transitions[ 6 ].Condition = x2;
a1.Transitions[ 6 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 7, "2->0", 2, 0 ) );
a1.Transitions[ 7 ].Condition = x0;
a1.Transitions.Add( new Transition( 8, "2->1", 2, 1 ) );
a1.Transitions[ 8 ].Condition = !x0 & !x1 & !x2;
a1.Transitions[ 8 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 9, "2->2", 2, 2 ) );
a1.Transitions[ 9 ].Condition = x2;
a1.Transitions[ 9 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 10, "2->3", 2, 3 ) );
a1.Transitions[ 10 ].Condition = x1;
a1.Transitions[ 10 ].Actions.Add( z4 );
a1.Transitions.Add( new Transition( 11, "3->0", 3, 0 ) );
a1.Transitions[ 11 ].Condition = x0;
a1.Transitions[ 11 ].Actions.Add( z5 );
a1.Transitions.Add( new Transition( 12, "3->1", 3, 1 ) );
a1.Transitions[ 12 ].Condition = !x0 & !x1 & !x2;
a1.Transitions[ 12 ].Actions.Add( z6 );
a1.Transitions[ 12 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 13, "3->2", 3, 2 ) );
a1.Transitions[ 13 ].Condition = x2;
a1.Transitions[ 13 ].Actions.Add( z5 );
a1.Transitions[ 13 ].Actions.Add( z3 );
a1.Transitions.Add( new Transition( 14, "3->3", 3, 3 ) );
a1.Transitions[ 14 ].Condition = x1;
a1.Transitions[ 14 ].Actions.Add( z4 );

Automaton a2 = new Automaton( 2, "Подсчет слов" );
a2.InputVariables.Add( x0 );
a2.InputVariables.Add( x2 );
a2.OutputActions.Add( z7 );
a2.States.Add( new State( 0, "Основное состояние" ) );
a2.States.Add( new State( 1, "Слово" ) );
a2.Transitions.Add(
    new Transition( 0, "Петля на основном состоянии", 0, 0 ) );
a2.Transitions[ 0 ].Condition = x0 | x2;
```

```
            a2.Transitions.Add( new Transition( 1, "Слово началось", 0, 1 ) );
            a2.Transitions[ 1 ].Condition = !x0 & !x2;
            a2.Transitions[ 1 ].Actions.Add( z7 );
            a2.Transitions.Add( new Transition( 2, "Петля на слове", 1, 1 ) );
            a2.Transitions[ 2 ].Condition = !x2;
            a2.Transitions.Add( new Transition( 3, "Слово кончилось", 1, 0 ) );
            a2.Transitions[ 3 ].Condition = x2;

            Automaton a0 = new Automaton( 0, "Управление итерированием" );
            a0.InputVariables.Add( x0 );
            a0.OutputActions.Add( z2 );
            a0.OutputActions.Add( z9 );
            a0.States.Add( new State( 0, "Основное состояние" ) );
            a0.States.Add( new State( 1, "Процесс завершен" ) );
            a0.States[ 1 ].Actions.Add( z9 );
            a0.Transitions.Add(
                new Transition( 0, "Петля на основном состоянии", 0, 0 ) );
            a0.Transitions[ 0 ].Condition = !x0;
            a0.Transitions[ 0 ].Actions.Add( a1 );
            a0.Transitions[ 0 ].Actions.Add( a2 );
            a0.Transitions[ 0 ].Actions.Add( z2 );
            a0.Transitions.Add(
                new Transition( 1, "Встреча нулевого символа", 0, 1 ) );
            a0.Transitions[ 1 ].Condition = x0;
            a0.Transitions[ 1 ].Actions.Add( a1 );
            a0.Transitions[ 1 ].Actions.Add( a2 );

            am.Automatons.Add( a0 );
            am.Automatons.Add( a1 );
            am.Automatons.Add( a2 );

            SoapFormatter formatter = new SoapFormatter();
            Stream stream = File.Create( "AutomatonManager.soap" );
            formatter.Serialize( stream, am );
            stream.Close();
        }
    }
}
```

П3.2. LoCoL_Demo

П3.2.1. Form1.cs

```
// Copyright:  (C) Alexander S. Naumov aka GooRoo, 2004.
// E-mail:     gooroo@bk.ru
// Last update:23/05/2004

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;
using System.Runtime.Serialization.Formatters.Soap;

using LoCoL;
using LoCoL.Logs;
```

```
namespace LoCoL_Demo
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 :
        System.Windows.Forms.Form, IAutomatonInput, IAutomatonOutput
    {
        private System.Windows.Forms.TextBox textBoxInput;
        private System.Windows.Forms.TextBox textBoxOutput;
        private System.Windows.Forms.Button buttonRun;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Timer timerLogUpdate;
        private System.Windows.Forms.TextBox textBoxBuffer;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.ComponentModel.IContainer components;

        public Form1()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.textBoxInput = new System.Windows.Forms.TextBox();
            this.textBoxOutput = new System.Windows.Forms.TextBox();
            this.buttonRun = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.timerLogUpdate = new System.Windows.Forms.Timer(this.components);
            this.textBoxBuffer = new System.Windows.Forms.TextBox();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.groupBox4 = new System.Windows.Forms.GroupBox();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
            this.groupBox3.SuspendLayout();
```

```
this.groupBox4.SuspendLayout();
this.SuspendLayout();
//
// textBoxInput
//
this.textBoxInput.BorderStyle =
    System.Windows.Forms.BorderStyle.FixedSingle;
this.textBoxInput.Location = new System.Drawing.Point(8, 16);
this.textBoxInput.Name = "textBoxInput";
this.textBoxInput.Size = new System.Drawing.Size(144, 20);
this.textBoxInput.TabIndex = 0;
this.textBoxInput.Text = "abc 123 12a";
//
// textBoxOutput
//
this.textBoxOutput.BorderStyle =
    System.Windows.Forms.BorderStyle.FixedSingle;
this.textBoxOutput.Location = new System.Drawing.Point(8, 16);
this.textBoxOutput.Name = "textBoxOutput";
this.textBoxOutput.ReadOnly = true;
this.textBoxOutput.Size = new System.Drawing.Size(144, 20);
this.textBoxOutput.TabIndex = 1;
this.textBoxOutput.Text = "";
//
// buttonRun
//
this.buttonRun.Cursor = System.Windows.Forms.Cursors.Hand;
this.buttonRun.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
this.buttonRun.Location = new System.Drawing.Point(8, 176);
this.buttonRun.Name = "buttonRun";
this.buttonRun.Size = new System.Drawing.Size(160, 23);
this.buttonRun.TabIndex = 2;
this.buttonRun.Text = "&Пуск";
this.buttonRun.Click += new System.EventHandler(this.buttonRun_Click);
//
// textBox1
//
this.textBox1.Anchor =
    ((System.Windows.Forms.AnchorStyles)
    ((((System.Windows.Forms.AnchorStyles.Top
    | System.Windows.Forms.AnchorStyles.Bottom)
    | System.Windows.Forms.AnchorStyles.Left)
    | System.Windows.Forms.AnchorStyles.Right)));
this.textBox1.BorderStyle = System.Windows.Forms.BorderStyle.None;
this.textBox1.Location = new System.Drawing.Point(8, 16);
this.textBox1.Multiline = true;
this.textBox1.Name = "textBox1";
this.textBox1.ReadOnly = true;
this.textBox1.ScrollBars = System.Windows.Forms.ScrollBars.Both;
this.textBox1.Size = new System.Drawing.Size(608, 168);
this.textBox1.TabIndex = 3;
this.textBox1.Text = "";
this.textBox1.WordWrap = false;
//
// timerLogUpdate
//
this.timerLogUpdate.Enabled = true;
this.timerLogUpdate.Tick +=
    new System.EventHandler(this.timerLogUpdate_Tick);
//
// textBoxBuffer
```

```
//
this.textBoxBuffer.BorderStyle =
    System.Windows.Forms.BorderStyle.FixedSingle;
this.textBoxBuffer.Location = new System.Drawing.Point(8, 16);
this.textBoxBuffer.Name = "textBoxBuffer";
this.textBoxBuffer.ReadOnly = true;
this.textBoxBuffer.Size = new System.Drawing.Size(144, 20);
this.textBoxBuffer.TabIndex = 1;
this.textBoxBuffer.Text = "";
//
// groupBox1
//
this.groupBox1.Controls.Add(this.textBoxInput);
this.groupBox1.Location = new System.Drawing.Point(8, 8);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(160, 48);
this.groupBox1.TabIndex = 4;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Входная строка";
//
// groupBox2
//
this.groupBox2.Controls.Add(this.textBoxOutput);
this.groupBox2.Location = new System.Drawing.Point(8, 64);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(160, 48);
this.groupBox2.TabIndex = 4;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Выходная строка";
//
// groupBox3
//
this.groupBox3.Controls.Add(this.textBoxBuffer);
this.groupBox3.Location = new System.Drawing.Point(8, 120);
this.groupBox3.Name = "groupBox3";
this.groupBox3.Size = new System.Drawing.Size(160, 48);
this.groupBox3.TabIndex = 4;
this.groupBox3.TabStop = false;
this.groupBox3.Text = "Буфер";
//
// groupBox4
//
this.groupBox4.Anchor =
    ((System.Windows.Forms.AnchorStyles)
    ((((System.Windows.Forms.AnchorStyles.Top
    | System.Windows.Forms.AnchorStyles.Bottom)
    | System.Windows.Forms.AnchorStyles.Left)
    | System.Windows.Forms.AnchorStyles.Right)));
this.groupBox4.Controls.Add(this.textBox1);
this.groupBox4.Location = new System.Drawing.Point(176, 8);
this.groupBox4.Name = "groupBox4";
this.groupBox4.Size = new System.Drawing.Size(624, 192);
this.groupBox4.TabIndex = 5;
this.groupBox4.TabStop = false;
this.groupBox4.Text = "Протокол";
//
// Form1
//
this.AcceptButton = this.buttonRun;
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(808, 205);
```

```
        this.Controls.Add(this.groupBox4);
        this.Controls.Add(this.groupBox1);
        this.Controls.Add(this.buttonRun);
        this.Controls.Add(this.groupBox2);
        this.Controls.Add(this.groupBox3);
        this.Name = "Form1";
        this.Text = "LoCoL Demo";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.groupBox1.ResumeLayout(false);
        this.groupBox2.ResumeLayout(false);
        this.groupBox3.ResumeLayout(false);
        this.groupBox4.ResumeLayout(false);
        this.ResumeLayout(false);

    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }

    private AutomatonManager m_Manager;
    private int m_MangerTimerID;
    private StringBuilder m_StringBuilder;
    private int m_WordCount;

    private void Form1_Load(object sender, System.EventArgs e)
    {
        SoapFormatter formatter = new SoapFormatter();
        m_Manager = (AutomatonManager)formatter.Deserialize(
            File.OpenRead( "AutomatonManager.soap" ) );
        m_Manager.Automatons[ 0 ].RegisterInput( this );
        m_Manager.Automatons[ 0 ].RegisterOutput( this );
        m_Manager.Automatons[ 1 ].RegisterInput( this );
        m_Manager.Automatons[ 1 ].RegisterOutput( this );
        m_Manager.Automatons[ 2 ].RegisterInput( this );
        m_Manager.Automatons[ 2 ].RegisterOutput( this );
        m_StringBuilder = new StringBuilder();

        m_Manager.RegisterLogger( new TextWriteLogger(
            new StringWriter( m_StringBuilder ) ) );
    }

    private void buttonRun_Click(object sender, System.EventArgs e)
    {
        textBoxInput.Enabled = false;
        textBoxBuffer.Enabled = false;
        textBoxOutput.Enabled = false;
        textBoxWordCount.Enabled = false;
        textBoxOutput.Text = "";
        textBoxBuffer.Text = "";
        textBoxWordCount.Text = "0";
        m_WordCount = 0;
        buttonRun.Enabled = false;
        m_Manager.Reset();
        m_MangerTimerID = m_Manager.SetEventTimer( new int[] { 0 }, 1000 );
```

```
}

[InputVariable( 0 )]
public bool X0()
{
    return textBoxInput.Text.Length == 0;
}

[InputVariable( 1 )]
public bool X1()
{
    return !X0() && char.IsDigit( textBoxInput.Text[ 0 ] );
}

[InputVariable( 2 )]
public bool X2()
{
    return !X0() && char.IsSeparator( textBoxInput.Text[ 0 ] );
}

[OutputAction( 2 )]
public void Z2()
{
    textBoxInput.Text = textBoxInput.Text.Remove( 0, 1 );
}

[OutputAction( 3 )]
public void Z3()
{
    textBoxOutput.Text += textBoxInput.Text[ 0 ];
}

[OutputAction( 4 )]
public void Z4()
{
    textBoxBuffer.Text += textBoxInput.Text[ 0 ];
}

[OutputAction( 5 )]
public void Z5()
{
    for ( int i = textBoxBuffer.Text.Length-1; i >= 0; i-- )
    {
        textBoxOutput.Text += textBoxBuffer.Text[ i ];
    }
    textBoxBuffer.Text = "";
}

[OutputAction( 6 )]
public void Z6()
{
    textBoxOutput.Text += textBoxBuffer.Text;
    textBoxBuffer.Text = "";
}

[OutputAction( 7 )]
public void Z7()
{
    m_WordCount++;
    textBoxWordCount.Text = m_WordCount.ToString();
}
```

```
        [OutputAction( 9 )]
        public void Z9()
        {
            m_Manager.KillTimer( m_MangerTimerID );
            buttonRun.Enabled = true;
            textBoxInput.Enabled = true;
            textBoxBuffer.Enabled = true;
            textBoxOutput.Enabled = true;
            textBoxWordCount.Enabled = true;
        }

        private string m_LogString;

        private void timerLogUpdate_Tick( object sender, System.EventArgs e )
        {
            string str = m_StringBuilder.ToString();
            if ( !str.Equals( m_LogString ) )
            {
                textBox1.Text = str;
                textBox1.Select( str.Length, 0 );
                textBox1.ScrollToCaret();
                m_LogString = str;
            }
        }
    }
}
```