Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

Д.И. Суясов, А.А. Шалыто

Автоматическое документирование программных проектов на основе автоматного подхода

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2006

Оглавление

ВВЕДЕНИЕ

Сегодня на передний план при создании информационных ресурсов выходит индивидуальность пользователя и скорость их разработки. Именно на это ориентируются большинство создателей идей и решений во всех сферах человеческой жизнедеятельности. Это свойственно также и производству программного обеспечения (ПО). Индивидуальность пользователя заставляет производителя создавать все время новые и более совершенные продукты, а ускоряющийся темп приводит к необходимости закрывать глаза на очень важные особенности при производстве ПО. Одной из таких особенностей является отношение к разработке документации на создаваемое ПО [1].

Документация в этом случае может восприниматься по-разному. Для программистов, документацией, к примеру, может являться правила использования того или иного языка программирования или шаблон, на основе которого они должны создавать свои программы. Для пользователя, информация о программе, о лицензионных условиях – тоже документация. В любом случае, документ, объясняющий или описывающий продукт, будет полезен, а в некоторых случаях необходим для большинства пользователей программ.

В виду ограниченных сроков создания новых программных продуктов, зачастую, они плохо документируются. Автоматическое документирование могло бы помочь и избавить создателя программного продукта от некоторой части работы. Суть автоматического документирования заложена в первом слове «автоматическое»: среда разработки «сама создает» документацию или шаблон будущей документации на основе реализуемого проекта. Это не заменяет полную проектную документацию, но может упростить процесс ее создания или поможет дополнить ее.

Различными компаниями реализован ряд документаторов. К примеру, компанией *Sun* создан автоматический документатор *JavaDoc* [2]. Его основной недостаток – программа документирует программные проекты, реализованные только на языке программирования *Java*. Кроме того, он не оставляет пользователю возможности влиять на процесс работы, а итоговый вид документации по формату и стилю получается всегда одним и тем же. Аналогично, существуют и другие программы, но они также выполняют весьма узкие задачи.

Обобщить и развить идеи автоматического документирования может помочь автоматный подход [3]. Он весьма прост в понимании. Достоинства подхода позволили упростить разработку и реализацию данного проекта на всех этапах, и, в то же время, при создании проекта не привели к ограничению в средствах реализации. В работе описаны три метода создания документации, каждый из которых основывается на автоматном подходе.

Различия реализованных методов заключаются в подходе к решению задачи. В первом методе применяются «крупные» автоматы, которые по этой причине весьма трудно повторно использовать, во втором – этот недостаток устранен, а третий метод – модификация второго, основанная на классических подходах к построению компиляторов.

Перечислим основные идеи, применяемые при выполнении задачи автоматического документирования.

1. Внешний вид документации должен различаться для различных форматов данных, что определяется целями, стоящими перед пользователем. Например, если для языка разметки *HTML* существенной информацией будет являться структура страницы (страницы сайта), то для языка сценариев *JavaScript* важно знать задачи, выполняемые отдельными функциями.

2. Синтаксис отдельных документируемых файлов существенно отличается друг от друга. Необходимо разработать общий принцип анализа применяемых форматов.

3. Появление новых форматов хранения информации и исходного кода языков программирования требует обеспечения возможности для оперативного добавления новых функций для анализа этих форматов без изменения принятой идеи автоматического документирования. Таким образом, описание форматов не должно включаться в систему документирования, а лишь загружаться из внешнего источника в процессе работы.

4. Эта система должна быть устойчивой к ошибкам, возникающим как при анализе документируемых файлов, так и при создании файлов документации. Она должна указать пользователю на ошибку. Это одновременно упрощает процесс реализации системы, так как любая нештатная ситуация будет мгновенно отслеживаться, и позволяет пользователям системы самостоятельно определять неполадки и устранять их.

Цель настоящей работы – разработка методов автоматического документирования программных проектов на основе автоматного подхода с учетом различий в форматах представления данных, входящих в состав проектов. Методы должны быть максимально универсальными и ошибкоустойчивыми, предоставлять документацию в удобной и доступной форме и позволять достаточно просто добавлять новые функциональные возможности.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

1.1. Обоснование необходимости автоматического документирования

Программный проект состоит не только из набора исходных файлов, созданных на основе синтаксиса используемого языка программирования, но и из файлов мультимедиа (картинки, видео, звук), текстовых файлов, файлов настроек и т.д.

Типы файлов, входящих в состав проекта, можно разделить на группы.

1. Статические данные (текст), которые содержат основную информацию проекта – исходный текст программы. Информация хранится в файлах с расширениями «`.html`», «`.htm`», «`.xml`» в случае Интернет проектов, и в файлах с расширением «`.java`» для проектов на языке программирования *Java*.

2. Графические и звуковые данные дополняют проекты. Они хранятся в отдельных файлах в форматах, таких как *JPEG*, *GIF*, *MPEG*, *BMP*, *MP3* и т.д. Текстовый формат мультимедиа данных не несет информации для человека.

3. *SQL, JavaScript* и другие сценарные языки входят в состав исходных текстов, но при этом их синтаксис отличается от основного текста файлов, в которые они встроены.

В результате проектирования и разработки программного проекта он может резко увеличиться. При этом количество файлов не позволит динамично находить требуемую информацию, требуемый документ или файл. Это затрагивает, прежде всего, разработчиков программного проекта. Значительно упростить работу может помочь проектная документация [1], описывающая концепцию и состав проекта. Автоматическое документирование, не являясь заменой проектной документации, а значительно дополняет ее.

Перечислим основные преимущества автоматической документации.

1. Документация создается автоматически на основе исходного кода проекта и комментариев разработчика (как, например, и в средстве *JavaDoc*). Это не требует дополнительных усилий при реализации.

2. Документация позволяет оперативно находить требуемый документ проекта без изучения исходных кодов.

3. Описание проекта можно проводить как с интеграцией в проект, так и отдельно от него. При этом в первом случае документация и файлы проектов могут иметь ссылки друг на друга.

4. Описание представляется в доступной форме, что помогает разобраться в проекте без углубления в реализацию.

Известны средства для автоматического документирования. Это, например, упомянутый выше, инструмент *JavaDoc* [2]. Это инструмент *DTDDoc* [4], документирующий набор *DTD* (*Document Type Definition* – определение типов документа) [5]. Кроме того, существует технология *XSLT* [6], преобразующая *XML*-файлы в другие представления, что является средством для представления хранящейся в метаданных *XML* информации.

1.2. Формальная постановка задачи

Основываясь на средстве *JavaDoc*, разработаем методы автоматического документирования программных проектов. Необходимо разработать основные принципы и методы обработки различных типов данных и создания документации. Следует сравнить предложенные методы между собой и с другими подходами и определить практическую пользу и перспективность данных методов в сфере автоматических обработчиков информации.

1.3. Требования к методу

Основываясь на формальной постановке задачи, нужно выделить моменты, которые будут являться основой при создании методов автоматического документирования.

Разработанные методы должны учитывать состав проекта. Любой тип файла, являющийся составной частью проекта, должен быть воспринят ими вне зависимости от категории, к которой он относится.

Методы призваны создать интуитивно понятную для непосвященного человека документацию, предоставлять возможность изменения ее внешнего вида и определения информации, которая должна быть документирована.

Они должны уметь анализировать статические данные проекта и программные элементы на основе комментариев разработчика в исходных кодах, имеющих специальный вид. Для этого необходимо разделить элементы проекта (файлы) на функциональные части – блоки, которые станут основами при создании документации. Анализ различных блоков должен осуществляться независимо друг от друга.

Неотъемлемым свойством разрабатываемых методов является расширяемость. Оно позволяет добавлять новые возможности пользователю в поиске информации в проектных файлах, определении новых форматов файлов для анализа с дальнейшим внесением найденной информации в файлы документации.

Появление ошибок в процессе работы пользователя по вине самого пользователя или разработчика не должно сказываться на работоспособности методов. Нужно предоставить максимальную возможность по самостоятельному исправлению пользовательских ошибок.

ГЛАВА 2. РАЗБОР ЗАДАЧИ

2.1. Схема системы

В предыдущей главе была приведена формулировка задачи по построению системы автоматического документирования программных проектов. Эту систему можно представить в виде диаграммы, в основе которой лежат промежуточные типы хранения данных и процессы взаимодействия между ними. На рис. 1 приведена упрощенная схема подобной системы.

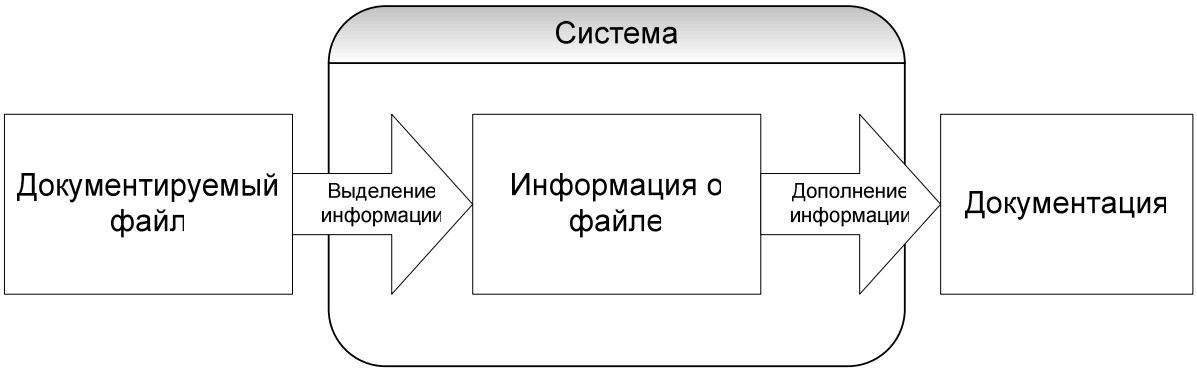

Рис. 1. Схема процесса документирования

Перечислим составляющие приведенной схемы.

1. Документируемый файл – объект, по отношению к которому и для которого создается документация.

2. Выделение информации – процесс изъятия из файла информации, которая в дальнейшем будет добавлена в текст документации к нему.

3. Информация о файле – промежуточный формат хранения выделенной информации в системе.

4. Дополнение информации – процесс подстановки выделенной из документируемого файла информации в файлы в соответствии с элементами

форматирования. Этот процесс позволяет представить информацию в том виде, в котором она необходимо конечному пользователю.

5. Документация – конечный формат представления данных, отображаемый пользователю системы.

Как следует из схемы, при ее построении было сделано предположение, что процесс выделения информации и процесс собственно создания документации отделены друг от друга и не действуют одновременно.

2.2. Ограничения документируемого файла

Распознавание различных форматов файлов, находящихся в составе программных проектов, входит в задачи системы документирования.

Структура и содержимое файлов могут сильно различаться. Это могут быть HTML, XML файлы, а также другие файлы, написанные на разных языках разметки. Пример подобного текста на языке разметки HTML приведен на рис. 2.

```
<html>
  <head>
    <title> Заголовок страницы </title>
    <link rel=”stylesheet” type=”text/css” href=”stylesheet.css”>
  </head>
  <body>
    <p class=”head”>
      Заголовок
    </p>
    <p class=”content”>
      Текст
    </p>
  </body>
</html>
```

Рис. 2. Пример *HTML* файла

Кроме языков разметки в состав проекта могут входить другие структурированные форматы[1]. Проект может содержать файлы таких языков программирования, как *JavaScript*, *PHP*, *Perl*, *Java* и т.д. Он может содержать также таблицы стилей, например, *CSS*, *XSL*.

Для человека выделение информации из структурированных файлов обычно не составляет большого труда, в особенности, если человек знаком с форматом представления данных в этих файлах. Аналогично, для автоматического распознавания данных в подобных файлах необходимо «научить» систему понимать различные форматы.

Сложности по выделению информации возникают при работе с такими форматами данных, как картинки, файлы видео, звука, мультимедиа и т.д. О таких данных нельзя сказать ничего определенного при просмотре их текстового содержания. Единственные данные, определяемые из таких файлов – это название, дата создания файла, положение в структуре проекта, ссылки из других структурированных файлов и еще, может быть, другая информация.

Система документирования программных проектов должна определять все форматы файлов и в соответствии со знаниями об этих форматах документировать их.

2.3. Процесс выделения информации

Файлы в составе проекта содержат полезную информацию для пользователя системы, которая в дальнейшем будет каким-либо образом

[1] В документе структурированный формат – это формат данных, для которого четко сформулированы правила, по которым эти данные составлены, и которые могут быть использованы человеком напрямую (не бинарный вид). Например, считается, что *HTML*-файл имеет структурированный формат, а *GIF*, *JPG* файлы – неструктурированный формат.

отражена в документации к проекту. Процесс выборки этой информации является первоочередной задачей создаваемой системы документирования.

Документируемые файлы могут иметь самую разнообразную структуру и вид. В процессе выборки информации из текста этих файлов система должна:

- распознать формат представления данных;
- найти положение и объем интересующей информации;
- преобразовать полученную информацию в промежуточный формат хранения внутри системы.

Распознавание формата файла можно автоматически осуществить исходя из расширения файла. Например, если файл имеет расширение `.html`, то и формат у него – *HTML*.

После распознавания формата следует поиск информации. На этой стадии необходимы некоторые инструкции от пользователя: система должна знать, какую именно информацию хочет видеть пользователь в конечной документации. Указанные инструкции должны быть изложены и доступны системе до начала ее работы. Это необходимо для унификации выборки информации для разных файлов одного формата, и в то же время для выполнения условия автоматичности документирования. Правила могут быть сформулированы пользователем и интерпретируемы системой различным образом, что будет показано в разд. 4. Определение объема информации также требует инструкций пользователя, которые формулируются до начала работы системы.

Процесс выделения информации, кроме всего прочего, предполагает также хранение выделенной информации перед началом создания документации. Без данного условия, с одной стороны, можно было бы создать параллельные процессы выделения информации и их

документирования, но, с другой стороны, это ограничивало бы возможность использования информации.

Для описанного в разд. 2.2 случая неструктурированных форматов данных (бинарных файлов) или форматов, для которых инструкции по выделению информации не определены, второй элемент из списка, приведенного выше, (поиск информации внутри файла) упускается. Единственные данные, определяемые из этих файлов – их характеристики, предоставляемые операционной системой.

2.4. Промежуточный формат хранения данных

Промежуточный формат хранения данных, как было сказано выше, позволяет не ограничивать систему в пользовании выделенной из документируемых файлов информацией – система сможет, к примеру, одновременно использовать данные, найденные в разных частях файла. Условие, которое должно выполняться для промежуточного формата, – легкость доступа системы к хранимой информации, что позволит гибко использовать эту информацию. Выполнить это условие можно, если для блока информации определить имя.

Для каждого блока выделенной информации пользователь должен определить имя, по которому эта информация будет доступна. Если система нашла на основе некоторого одного правила несколько разных блоков информации, то различия имен определяется с помощью индекса.

К примеру, для приведенного на рис. 2 *HTML*-файла после выделения информации промежуточные данные могут быть представлены в табл. 1.

Таблица 1. Промежуточный формат данных для *HTML* файла, представленного на рис. 2.

Имя	Информация
title1	Заголовок страницы
stylesheet1	stylesheet.css
text1	Заголовок
text2	Текст

В табл. 1 индексы размещены после всех имен блоков информации, что требуется для унификации процесса.

2.5. Процесс создания документации

После выделения информации из документируемых файлов необходимо ее представить в виде, который требуется пользователю. Для этого следует часть этой информации совместить или изменить, выбрать конечный формат представления данных, добавить некоторые общие данные и записать все это по файлам.

Процесс создания документации предполагает, что система еще до начала работы будет знать структуру результирующих файлов. Шаблонный подход [7] позволяет добиться этого, а также предоставляет возможность пользователю участвовать в дизайне документации. Шаблоны со специальными командами внутри позволяют системе с легкостью манипулировать добытой в предыдущем процессе информацией.

Аналогично процессу выделения информации использование шаблонного подхода может быть реализовано по-разному. Различные подходы в реализации будут изложены в разд. 4.

2.6. Формат документации

Система должна в результате своей работы сформировать документацию ко всем файлам, поступающим на вход. Возможно существование нескольких вариантов структуры документации.

1. Для каждого обрабатываемого файла будет создан отдельный файл документации.

2. Документация будет состоять из набора файлов, не связанных с файлами проекта. Они создаются одни и те же для любой структуры проекта (такие файлы в дальнейшем будут называться общими).

3. Файлы будут создаваться как индивидуально для каждого обрабатываемого файла, так и для всей их совокупности в целом.

Последний вариант кажется наиболее приемлемым. В этом случае пользователь будет самостоятельно определять общие создаваемые файлы, а для каждого файла будут созданы документирующие файлы автоматически, и имена их будут соответствовать именам исходных файлов.

2.7. Дополненная схема процесса документирования

На рис. 3 изображена итоговая схема процесса документирования.

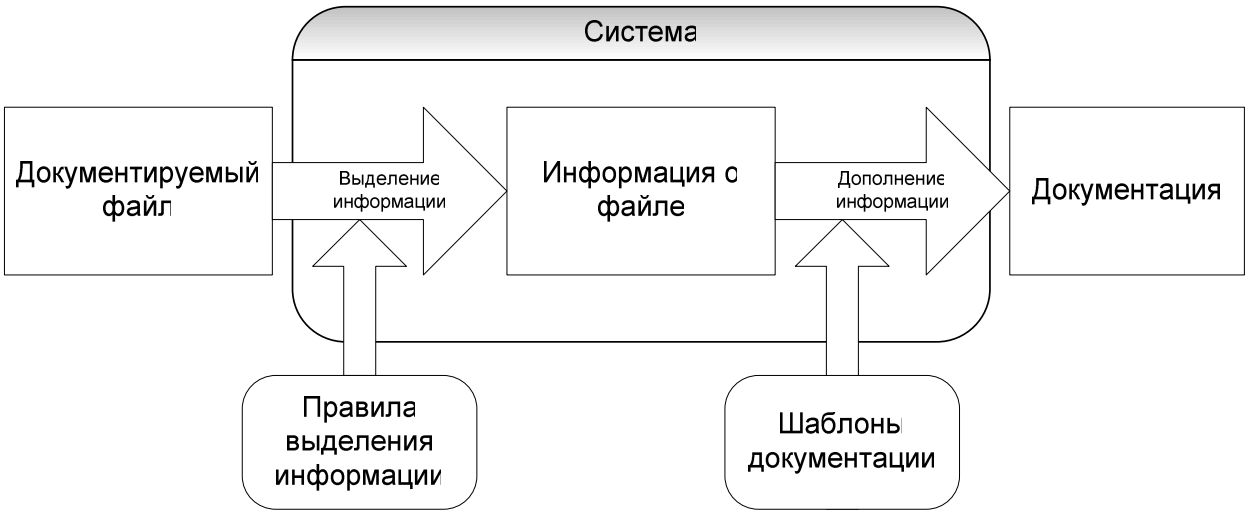

Рис. 3. Дополненная схема процесса документирования

Как видно из схемы, основными моментами процесса документирования с точки зрения создаваемой системы являются процессы выделения информации на основе определенных пользователем правил и дополнение шаблонов выделенной информацией.

ГЛАВА 3. АВТОМАТНЫЙ ПОДХОД В СТРУКТУРЕ МЕТОДА

3.1. Компоненты процессов системы

Перед системой автоматического документирования стоят две основные задачи: выделение информации из входных файлов и создание на ее основе документации – выходных файлов. Решение каждой задачи заключается в выполнении ряда действий.

Выделение информации можно представить как последовательность следующих действий.

1. Задание правил, по которым будет выделяться информация из входных файлов. Так как система должна иметь возможность документировать различные форматы файлов, то пользователь должен иметь возможность самостоятельно определять правила для каждого из форматов.

2. Преобразование полученных правил в некоторое внутреннее представление, которым можно было бы легко пользоваться в дальнейшем.

3. Загрузка документируемого файла и применение к нему преобразованного правила, определенного для формата этого файла.

4. Сохранение выделенной информации.

Из выделенной последовательности действий следует, что главные его составляющие (основа) – это формат начальных правил, которые система считывает из внешнего источника, и формат внутреннего представления правил, которые должны быть удобными для применения в выделении информации из документируемых файлов. Подробно процесс выделения информации представлен в виде схемы на рис. 4.

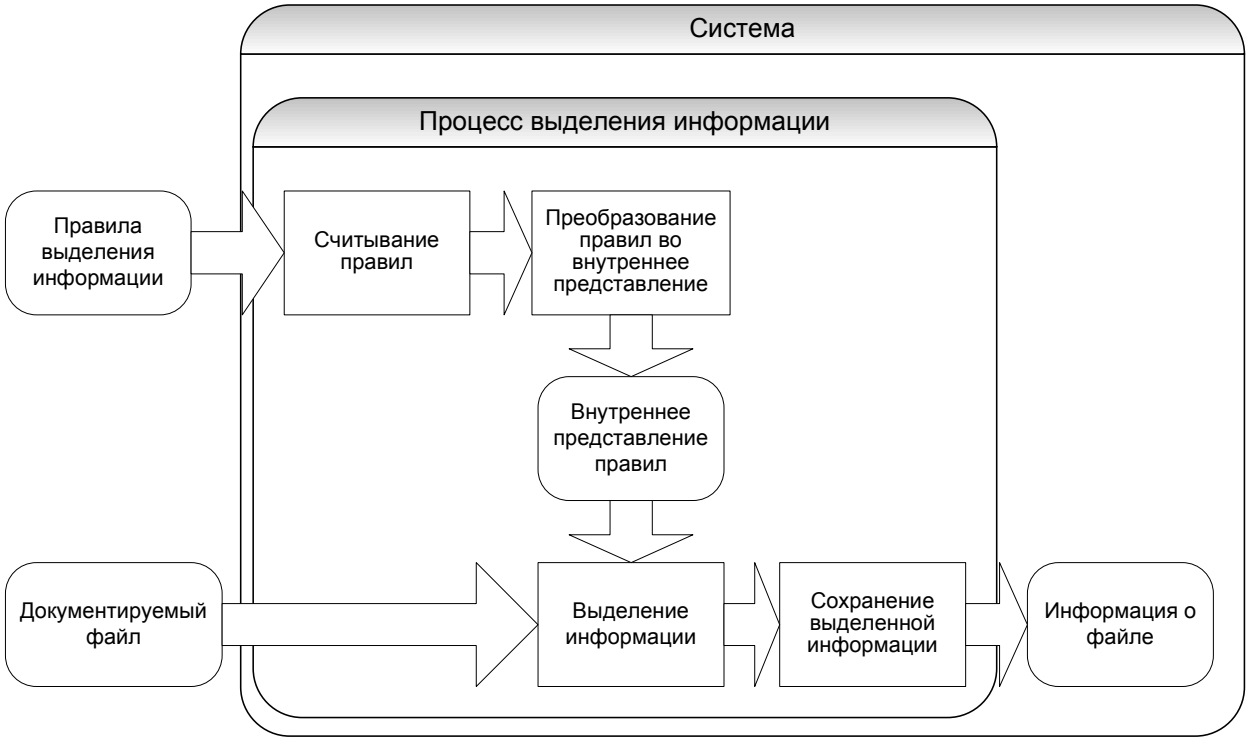

Рис. 4. Схема процесса выделения информации

Другой важной задачей системы является процесс документирования или заполнения заготовки информацией, выделенной на предыдущем этапе. Дизайн и содержание итоговой документации должны отражать предпочтения пользователя. Для этого было введено понятие шаблонов, загружаемых в систему извне.

Процесс документирования можно разделить на действия.

1. Система считывает текст шаблона из внешнего источника.

2. Текст шаблона преобразуется в структуру внутреннего представления. Это сделано для удобства манипуляции с элементами шаблона и добавления в него информации.

3. Информация о файле, которая была получена на предыдущем этапе, вставляется в загруженный шаблон. При этом должны выполняться все инструкции шаблона, которые обязаны отсутствовать в итоговой документации.

4. Итоговые документы сохраняются в указанном пользователем месте.

Схема процесса изображена на рис. 5.

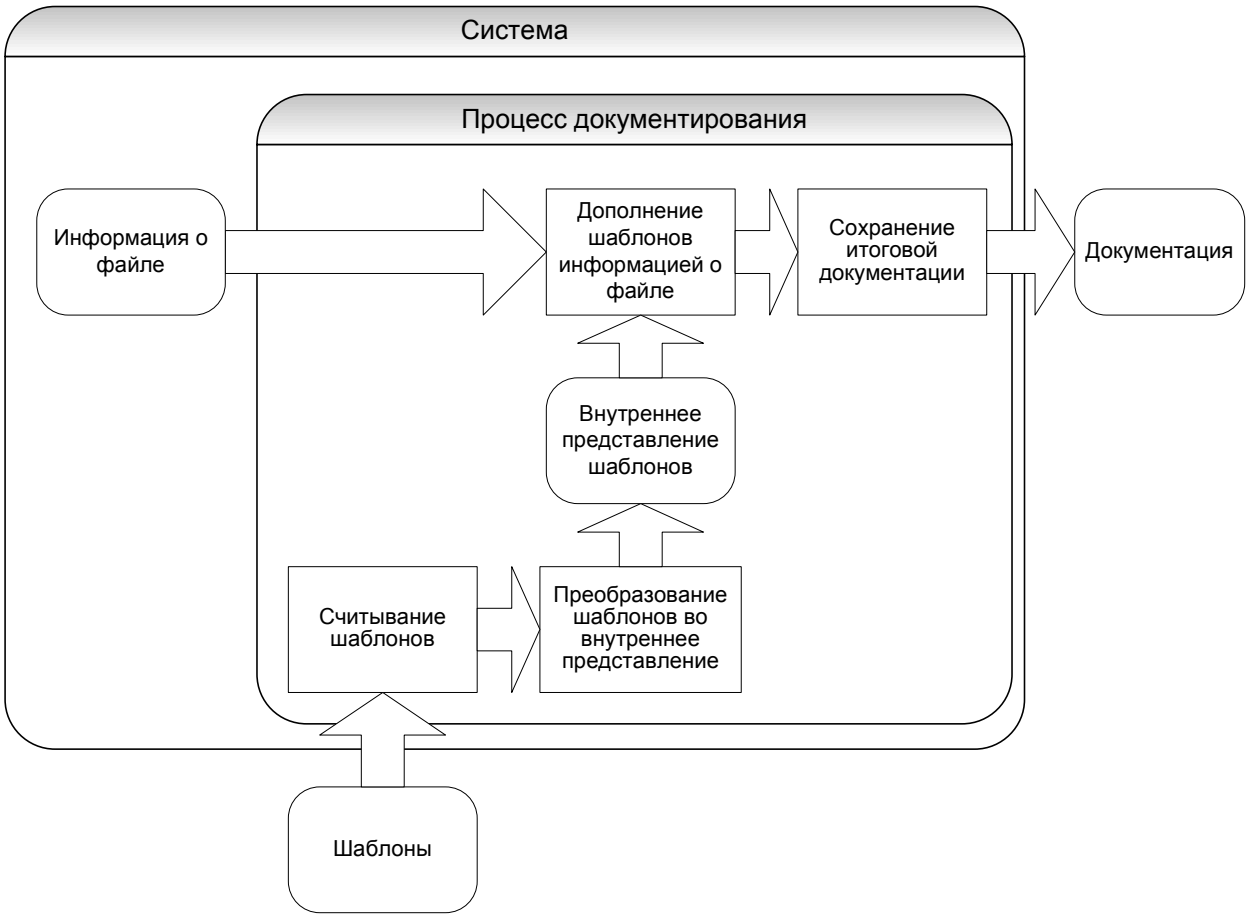

Рис. 5. Схема процесса документирования

3.2. Реализация компонентов на основе автоматного подхода

Теория автоматов основывается на понятии состояния [3, 8]. При переходе между состояниями происходит проверка переменных, которые в рамках разрабатываемой системы могут быть многокомпонентными, и выполняются действия.

Применение автоматного подхода в данной работе позволило достаточно быстро реализовать схему на рис. 3. Отметим основные преимущества, достигнутые на основе автоматного подхода.

1. Легкость и четкость разработки. Стало возможным без привязки к языку программирования и реализации разработать алгоритм, выполняющий поставленную задачу.

2. Без труда удалось разделить систему на отдельные компоненты.

3. В ходе разработки удалось максимально учесть все нюансы системы без обращения к языку программирования.

4. Реализация алгоритма после проектирования – изоморфное отображение спроектированных автоматов в программу.

5. Отладка в терминах автоматов существенно упростилась.

Процесс построения автоматов можно разделить на этапы.

1. Определяются задачи, реализуемые автоматом.

2. Выделяются состояния автоматов, которые описываются.

3. Строится граф переходов со связями между состояниями, определяются входные переменные, выходные воздействия, события, на которые реагирует автомат. Также строится схема связей автомата с источниками информации и объектами управления.

Для большей формализации разработки систем, основанных на автоматном подходе, была введена нотация графа переходов. Она во многом аналогична с используемой в работе «Проектирование программного обеспечения системы управления дизель-генераторами на основе автоматного подхода» [9]. Нотация для разрабатываемой системы приведена на рис. 6.

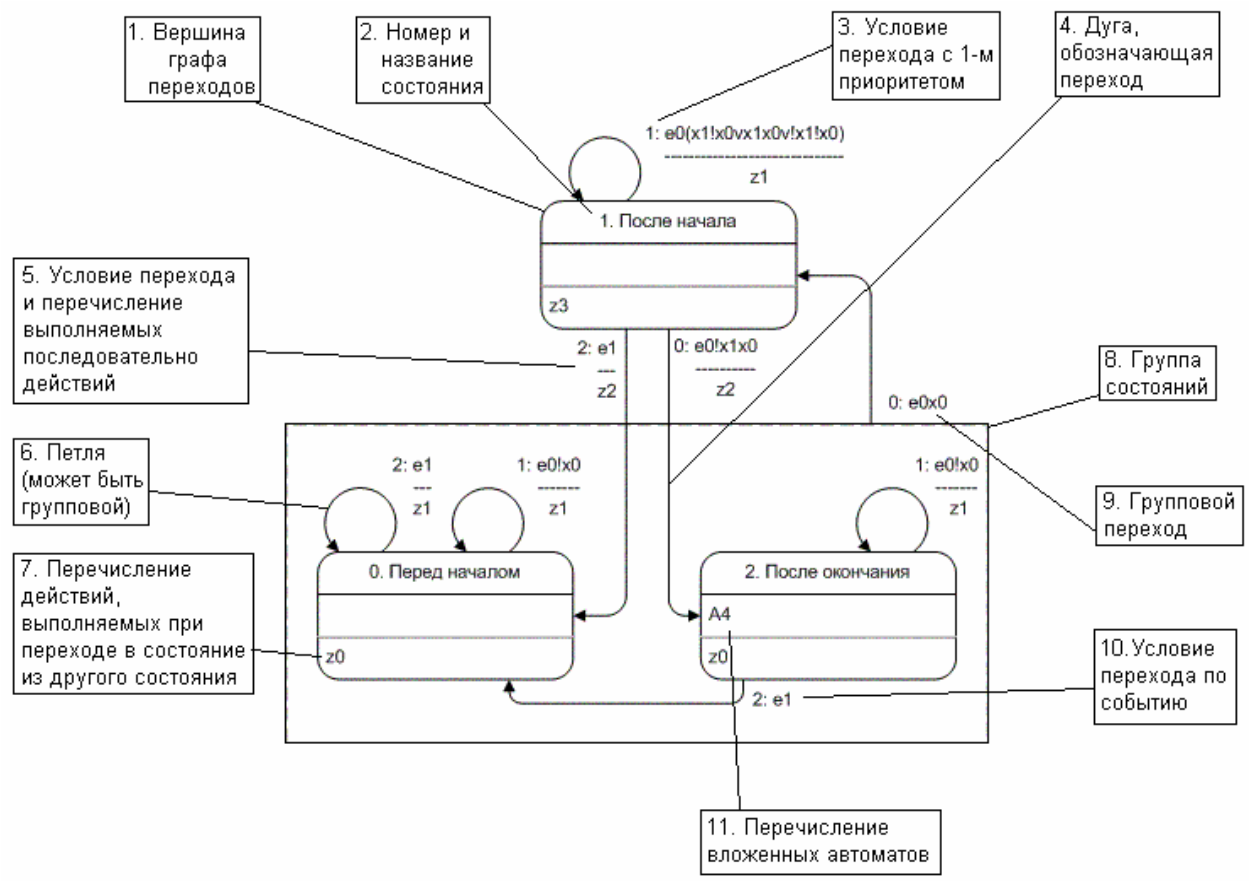

Рис. 6. Нотация графа переходов

Графы переходов, используемые в настоящей работе, описываются с использованием этой нотации.

ГЛАВА 4. МЕТОД НА ОСНОВЕ «ГИБКИХ» ПРАВИЛ

4.1. Общий принцип

Метод на основе «гибких» правил был разработан на основе схемы на рис.3. В качестве его основных составляющих (компонент системы) выступают конечные автоматы. Автоматы призваны решить основные задачи системы, к которым относятся перечисленные ниже.

1. Загрузка и анализ структуры документируемых файлов.
2. Анализ каждого файла проекта и поиск в нем комментариев специального вида, на основе которых выполняется поиск необходимой информации в тексте.
3. Выделение блоков информации из документируемого файла.
4. Занесение выделенной информации в шаблоны документации.
5. Создание дополнительных файлов документации.

Было спроектировано четыре конечных автомата, каждый из которых выполняет в системе определенную задачу.

Первый автомат управляет работой всех процессов, осуществляет анализ документируемых элементов проекта, выполняет последовательный запуск других автоматов, определяет ошибки и сообщает о них пользователю.

Второй автомат обеспечивает разбор структуры файла и выделяет блоки информации, необходимые для итоговой документации.

Третий автомат анализирует загруженный файл, выделяет информацию из него и запускает четвертый автомат.

Этот автомат помещает полученную предыдущими автоматами информацию в заготовленные шаблоны и создает содержимое файла документации.

Поиск информации в файлах выполняется на основе предварительно заданных правил, каждое из которых определяет начало и конец определенного блока информации при посимвольном считывании файлов. Все правила описаны вне системы и загружаются в формате *XML* [5].

Каждый файл, который должен быть проанализирован программой, делится разработчиком файла на смысловые блоки, которые разделяются между собой комментариями специального вида. Эти комментарии находятся в начале каждого блока. Поиском комментариев занимается автомат `A1` на основе правил, предварительно описанных в *XML*-формате. Для каждого блока задается собственный набор правил, которые будут использоваться при выделении из него информации.

Разные типы блоков имеют разные имена. Например, в рамках *HTML*-страницы могут встречаться блоки, выполняющие задачи навигации по сайту (меню), описания чего-либо или иллюстраций. Соответствующие им названия могут быть `menu`, `description` и `illustration`. Названия блоков описываются в *XML*-формате до начала работы системы. Поэтому встретив в документируемом файле комментарий следующего вида

`<!--@html ...-->`,

система определит `html` как имя существующего блока. Таким образом, определяются начало и название каждого нового блока.

Итоговый вид документации – результат обработки шаблонов, которые предварительно разрабатываются для каждого типа файла. Шаблоны – это документы, в которые в процессе работы системы будет добавлена информация, полученная при анализе документируемых файлов. Чаще всего формат шаблонов соответствует формату файлов итоговой документации (например, формат *HTML*). Заполненные шаблоны объединяются в файлы и записываются в каталог документации.

4.2. Описание формата правил

Любой формальный язык программирования, а именно на таком языке пишутся компоненты проектов, представляют собой набор правил [10]. Иногда, этот набор состоит из очень большого количества правил, досконально описывающих все тонкости языка.

При документировании файлов предполагается определять лишь часть информации, которая в дальнейшем понадобится при заполнении шаблона. Для выделения этой информации введено понятие «гибких» правил. Они помогут определить позиции в файле, на которых извлекаемая информация начинается и заканчивается.

В общем случае, для правил выделения информации можно ввести иерархическую систему. Рассмотрим пример по выделению заголовка *HTML*-файла из исходного текста. В этом случае, удобно разделить выделение фрагмента на два этапа, каждому из которых соответствует свое правило. На первом этапе выделяется заголовок вместе с обрамляющими тегами.

```
<title> Заголовок HTML файла </title>.
```

Это удобно сделать, так как тег `<title>` уникален на *HTML*-странице. Далее, применяя второе правило, выделяем непосредственно заголовок страницы.

```
Заголовок HTML файла.
```

В этом случае первое правило будем считать родительским по отношению ко второму.

Нужная информация может располагаться внутри сложной структуры файла, и для ее выделения понадобится большое количество условий. Поэтому правила для нахождения информации могут иметь сложный вид. В этом случае можно воспользоваться понятием булевой формулы [11].

Каждое правило будем соотносить с булевой формулой A^B^…^C, где «^» – знак конъюнкции, а операнды A, B, …, C – последовательности символов (в дальнейшем их будем называть словами). Символы должны идти один за другим для того, чтобы можно было считать правило выполненным. Порядок следования операндов в формуле играет решающую роль, так как он определяет порядок, в котором слова должны располагаться в тексте документируемого файла. Например, для того, чтобы определить в *HTML*-файле с текстом

```
...<title> Заголовок HTML файла</title>...
```

начало информации о заголовке страницы (*HTML*-страницы), необходимо, чтобы одна из формул была истиной:

A^C, A^B^C, A^B^B^C …,

где A – слово «<title», B – « » (пробел) и C – «>».

Таким образом, правила, определенные этими формулами, могут быть сформулированы следующим образом: «сначала в тексте должно идти слово <title, затем может быть любое количество пробелов до знака закрытия тега >». Для того чтобы определить окончание блока информации истинной должна быть одна из формул D^C или D^B, где D – слово «<\title», B и C – определенные выше символы пробела и «>» соответственно.

В каждой формуле помимо операндов A, B и C могут встречаться их отрицания ¬A, ¬B и ¬C. Отрицанием будем обозначать отсутствие заданного операндом A слова. Эта возможность помогает исключить ситуации с неверным выделением границ информации. К примеру, в языке сценариев *JavaScript* [9] при определении значения строковой переменной

```
...a = "Привет, \"МИР\"";...
```

начало определяемого фрагмента определяется с помощью формулы X, где X – символ «"», а окончание фрагмента – формулой ¬Y^X, где Y – символ «\». В примере метод корректно выделит фрагмент информации, несмотря на символы кавычек.

В булевой формуле позиция операнда имеет решающее значение. Она означает, что слова, которые представлены операндом, должны следовать после слов, которые представлены предыдущим операндом. Иногда могут возникать ситуации, в которых при проверке условия в заданной позиции текста должны располагаться любые слова кроме нескольких заданных. Для этого можно использовать несколько отрицаний. Например, при определении в тексте языка сценариев *JavaScript*

```
...  a = b;...
```

позиции первого значащего символа необходимо выполнения формулы ¬I^¬M^¬N, где I – символ пробела, M – символы перевода строки, N – символ табуляции.

Иными словами, символ будет считаться началом информационного фрагмента, если он не будет выбран – не будет являться пробелом, знаком табуляции или символом перевода строки (которые назовем условиями). Операнды I, M и N должны выполняться при анализе одного и того же символа. Но выше было введено правило, по которому позиция операнда в формуле задает позицию символа, тогда требуется заменить формулу выражением ¬P, где P – операнд, объединяющий введенные условия.

Некоторые формулы, входящие в одно правило, можно объединить. Например, бесконечную последовательность формул A^C, A^B^C, A^B^B^C,… можно заменить одной формулой вида A^*B^C, где знак «*» означает, что слово, определяемое операндом B, может встречаться в этом месте ноль и более раз. Аналогично можно ввести следующие знаки:

«!» – слово должно присутствовать ровно один раз;

«?» – слово может присутствовать один раз;

«+» – слово должно присутствовать не менее одного раза.

Формулы `!A^!B^!C` и `!A^!¬D^!C` могут быть также объединены в одну формулу `!A^!(B v ¬D)^!C`, где знак «v» – дизъюнкция. Такое объединение не противоречит введенному правилу о порядке следования операндов. В данном случае, каждая простая дизъюнкция, в которой используются только операнды, будет определять варианты слов, которые должны или могут присутствовать в заданной позиции.

Следуя введенным обозначениям, поиск границ блоков информации будет осуществляться на основе правил начала фрагмента информации и его окончания. Каждое из таких правил описывается по предложенной схеме. Можно определить блоки информации как вложенные друг в друга. Тогда проверка правила начала вложенного блока будет возможна только после выполнения правила начала родительского блока, а для завершения родительского блока необходимо принудительно завершить вложенный в него блок.

Все введенные правила описываются вне системы в формате *XML* [5]. Для того чтобы избежать проблем, связанных с ограничениями *XML* по использованию некоторых символов в тексте, каждое правило будем располагать в секции «`CDATA`» тегов «`<start>`» или «`<end>`». Внутри секции будут описываться правила вида:

```
!zero|*one,two,three|!ex(four&five),six|!seven
```

Данные правила эквивалентны введенным булевым формулам. В них символы логических операций заменены другими символами: символ «`|`» заменяет символ конъюнкции в аналогичной булевой формуле, символ «`,`» аналогичен дизъюнкции, а символы «!», «?», «*» и «+» определяют

количество повторений слов, заданных идущими после них операндами. Выражение «ex(…)» означает отрицание элементов, перечисленных в скобках.

4.3. Шаблонный подход

Система создает документацию на основе шаблонов [7]. Отличительной чертой каждого шаблона по сравнению с результирующим файлом документации, создаваемым на основе этого шаблона, является присутствие в тексте заглушек. Заглушка задает позицию в файле, в которую будет помещена определенная информация. Каждой заглушке соответствует имя, соотнесенное с этой информацией. В процессе работы системы все заглушки в шаблонах заменяются соответствующими фрагментами информации либо удаляются, если информация с определенным именем отсутствует. После замены или удаления отдельные шаблоны объединяются в готовые файлы документации.

Заглушка в шаблоне имеет следующий вид:

#@<имя блока>@<имя правила>;

где <имя блока> – название блока в документируемом файле, в котором должна находиться интересуемая информация, <имя правила> – название правила, на основе которого должна быть найдена эта информация.

При обработке документируемого файла могут возникать случаи, когда по одному правилу было найдено несколько разных фрагментов информации. Каждому из этих фрагментов будет присвоено одно и то же имя, которое соответствует имени правила. Во время заполнения шаблона вместо заглушки с аналогичным именем будет вставлен только первый найденный фрагмент. Для того чтобы добавить в шаблон все фрагменты с одним именем необходимо воспользоваться командой повтора. Эта команда имеет вид:

`#@repeat#<команда>; … #@repeatEnd#<команда>;`

где `<команда>` – имя правила, которое выделяет несколько фрагментов информации. Команда повтора ограничивает часть шаблона. Эта часть будет добавлена в результирующий файл несколько раз.

В шаблонах может присутствовать также команда условного оператора

`#@if#<команда>; … #@ifEnd#<команда>;`

где `<команда>` определяет имя правила. Если информация была найдена в документируемом файле на основе этого правила, то часть шаблона между началом и концом команды будет добавлена в результирующий файл документации.

4.4. Автоматная реализация

В методе на основе «гибких» правил было выделено четыре автомата. Первый автомат (A0) отвечает за общее функционирование системы, а остальные решают задачу автоматического документирования в рамках разработанной выше схемы (рис. 3). Итоговая схема системы представлена на рис. 7.

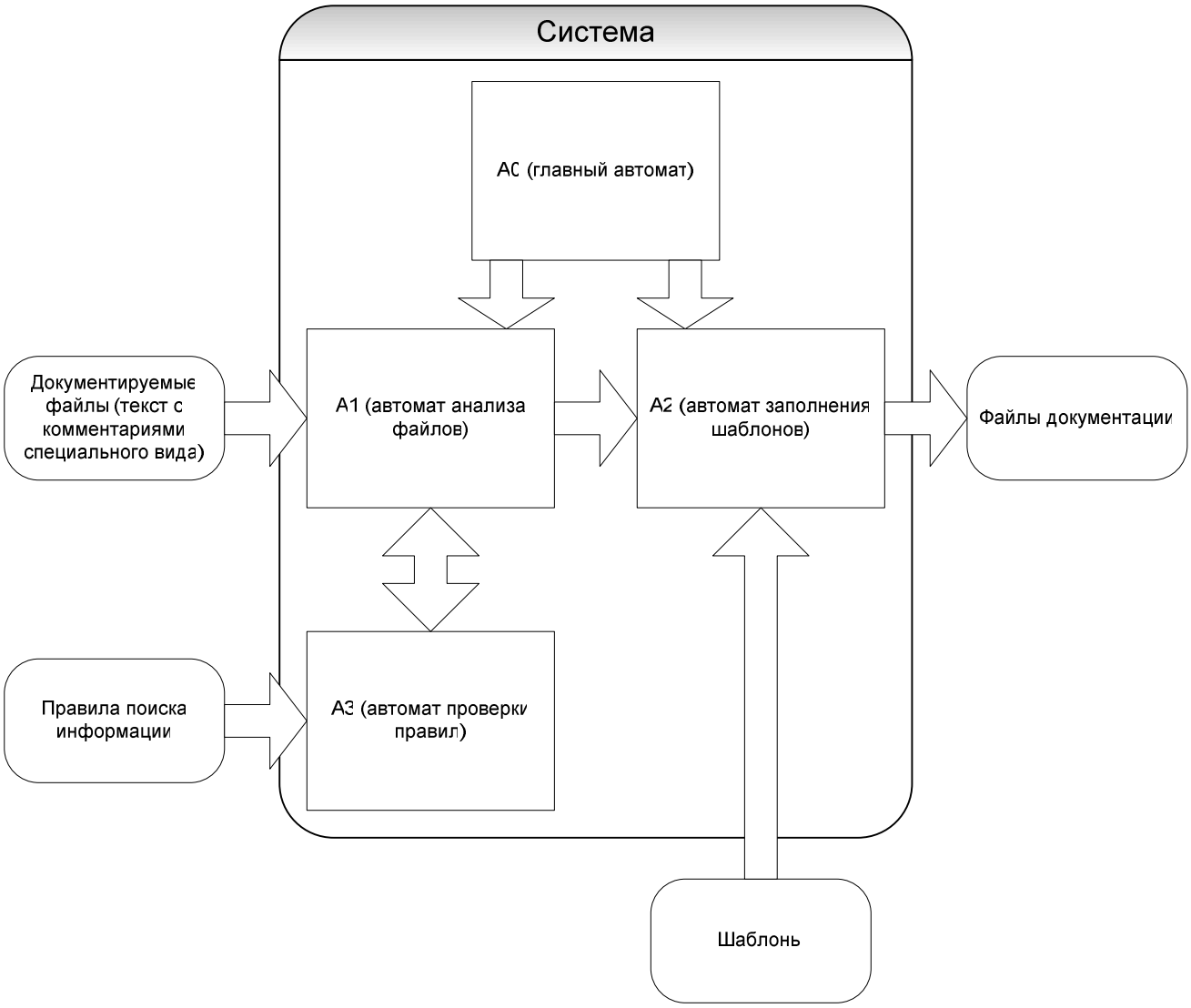

Рис. 7. Схема функционирования системы, основанной на «гибких» правилах

4.4.1. Описание автомата A0

Формальное описание

Автомат A0 управляет программой, формирует список файлов для обработки, запускает для каждого из них анализатор (автомат A1) либо автомат для заполнения шаблона A2, а также создает общие файлы документации.

Нумерация, перечень и описание состояний автомата A0

Автомат A0 осуществляет переход между состояниями, перечисленными в табл. 2.

Таблица 2. Состояния автомата A0

№	**Название состояния**	**Описание состояния**
0	Начало работы	Устанавливаются переменные, переданные командной строкой
1	Загрузка свойств	Загрузка файлов опций и шаблонов
2	Процесс документирования	Определяются список файлов для документирования и автомат-обработчик для каждого из них
3	Обработка файла и заполнение шаблона для него	Запускается автомат A1 для анализа документируемого файла
4	Заполнение шаблона	Запускается автомат A2 для заполнения шаблона по умолчанию для документируемого файла
5	Создание общих файлов	Определяет общие файлы документации
6	Заполнение шаблона для общего файла	Запускает автомат A2 для заполнения шаблона очередного общего файла
7	Завершение работы	Завершение работы программы

Перечень входных переменных автомата A0 и их описание

Ниже перечислены все входные переменные автомата A0.

x0 Массив ошибок не пуст

x1 Определено достаточно параметров для работы

x2 Перебраны все аргументы программы

x3 Существует необработанный файл

x4 Необходимо создать общий файл

x5 Существует шаблон для обрабатываемого файла

Перечень выходных воздействий автомата A0 и их описание

Ниже перечислены все выходные воздействия автомата A0.

z0 Завершает работу программы

z1 Обрабатывает аргументы командной строки

z2 Загружает файлы параметров программы и шаблонов

z3 Выводит в файл и на экран список ошибок

z4 Определяет неизвестные опции значениями по умолчанию

z5 Создает список файлов, подлежащих обработке

z6 Создает автомат для документирования очередного обрабатываемого файла

z7 Создает автомат для заполнения шаблона общего файла

z8 Инициализирует создание общих файлов

z9 Записывает результат автомата для заполнения шаблона общего файла в файл

z10 Создает автомат для заполнения шаблона по умолчанию для файла с

неизвестным расширением

z11 Записывает результат автомата для заполнения шаблона файла в файл

Схема связей автомата A0

Схема связей автомата A0 представлена на рис. 8.

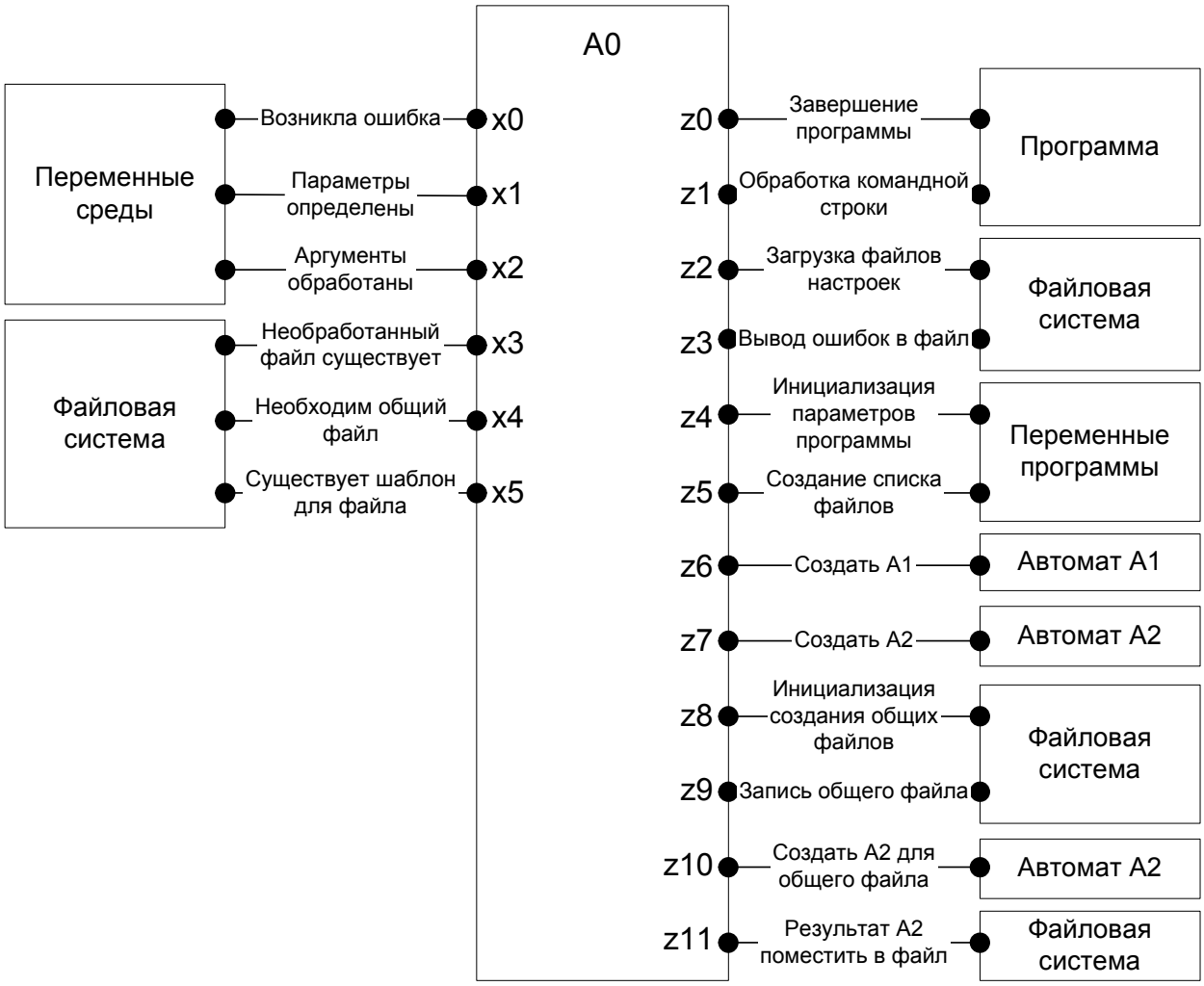

Рис. 8. Схема связей автомата A0

Граф переходов автомата A0

На рис. 9 представлен автомат переходов.

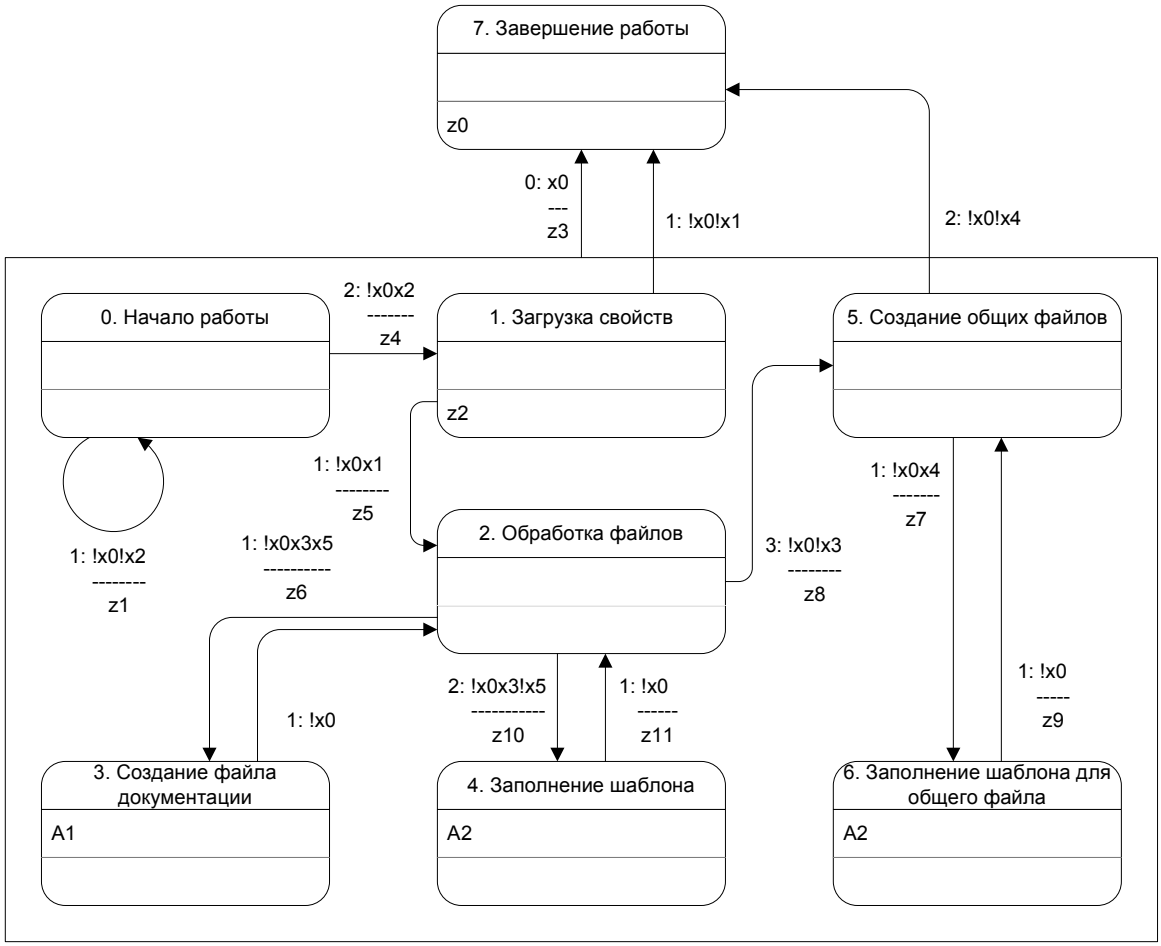

Рис. 9. Граф переходов автомата A0

4.4.2. Автомат анализа файлов A1

Формальное описание автомата A1

Автомат A1 анализирует документируемый файл, определяет описательные блоки (комментарии), выделяет информацию из текста файла, а также запускает автомат для заполнения шаблона выделенной информацией.

Нумерация, перечень и описание состояний автомата A1

В табл. 3 перечислены состояния автомата A1.

Таблица 3. Состояния автомата A1.

№	**Название состояния**	**Описание состояния**
0	Чтение файла	Загрузка файла проекта
1	Вне заголовка блока	Перебор символов файла, формирование событий для автомата A3, поиск специальных блоков – комментариев
2	Чтение имени атрибута	Определение имени свойства блока
3	Чтение значения атрибута	Определение значения свойства блока
4	Создание документации	Определение шаблонов для документации, их компоновка
5	Заполнение шаблона	Заполнение шаблона
6	Завершение работы автомата	Завершение работы автомата

Перечень входных переменных автомата A1 и их описание

Автомат A1 имеет десять входных переменных, перечисленных ниже.

x0 Программа выдала ошибку

x1 Начался описательный блок

x2 Файл закончился

x3 Выполнилось правило окончания блока

x4 Выполнилось подправило заданного правила

x5 Началось имя атрибута в заголовке блока

x6 Началось значение атрибута в заголовке блока

x7 Закончилось текущее правило

x8 Есть блок, для которого не заполнен шаблон

x9 Заполнен шаблон шапки документации файла

x10 Заполнен базовый шаблон файла

Перечень выходных воздействий автомата A1 и их описание

Выходные воздействия автомата A1 перечислены ниже.

z0 Чтение файла

z1 Загрузка очередного символа

z2 Завершение работы автомата A1

z3 Инициализация правил поиска информационных блоков

z4 Формирование события загрузки очередного символа

z5 Инициализация имени текущего атрибута значением по умолчанию

z6 Определение нового списка правил при поиске информационных блоков

z7 Сохранение значений атрибутов, выданных правилами

z8 Добавление символа к значению текущего атрибута

z9 Добавление символа к имени текущего атрибута

z10 Сохранение значения атрибутов нового блока, найденного в заголовке

z11 Удаление лишних символов из значений атрибутов

z12 Инициализация нового набора правил, являющихся подправилами текущего правила

z13 Создание файла документации

z14 Определение переменных для начала заполнения шаблона

z15 Создание автомата для заполнения шаблона

z16 Сохранение результата автомата заполнения шаблона

z17 Создание автомата для заполнения шапки автомата

z18 Создание автомата для заполнения базового шаблона

z19 Сохранение итогового текста документации в файл

z20 Сохранение данных для общих файлов документации

Схема связей автомата A1

Схема связей автомата A1 представлена на рис. 10.

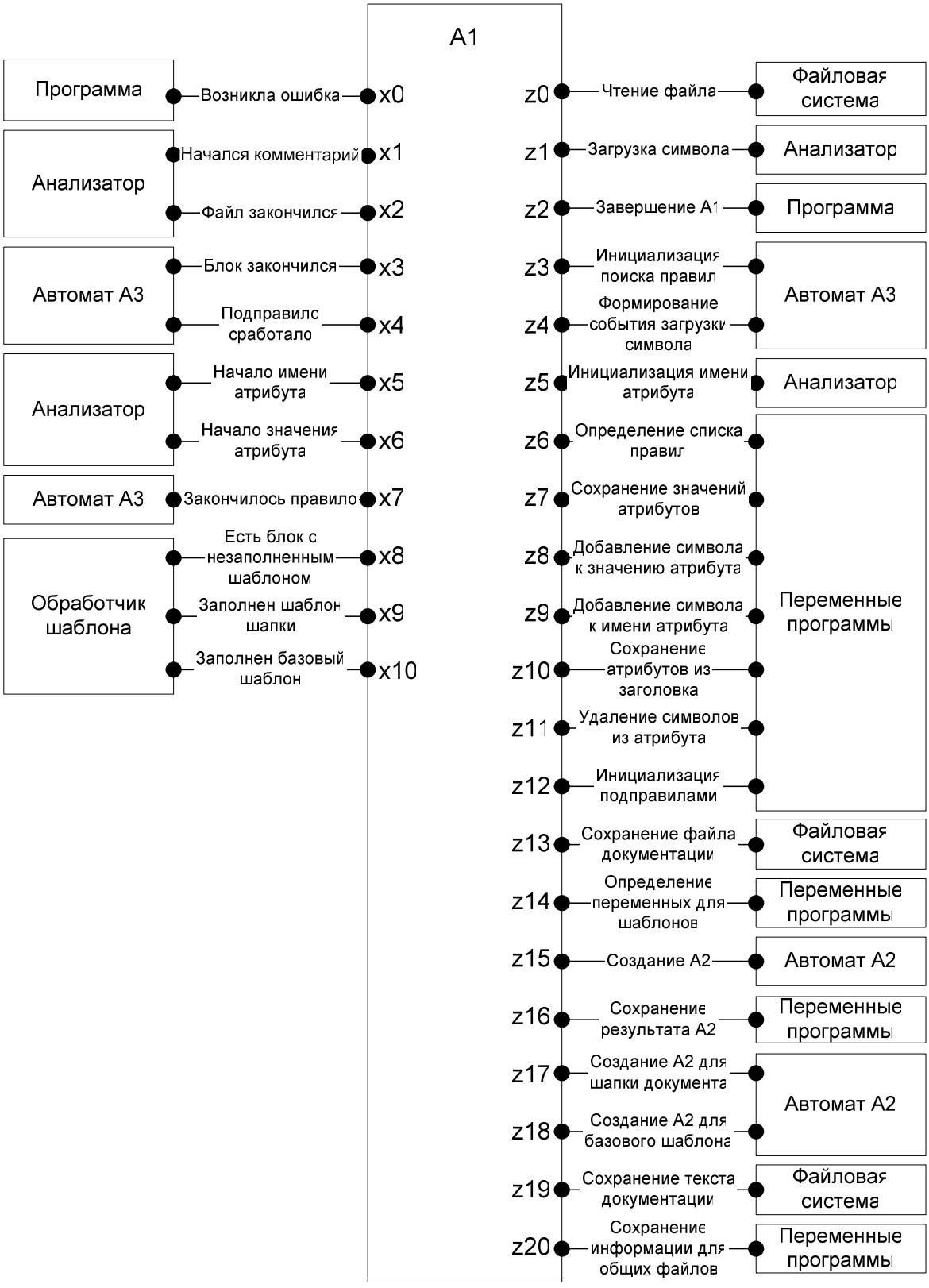

Рис. 10. Схема связей автомата A1

Граф переходов автомата A1

На рис. 11 приведен граф переходов автомата A1.

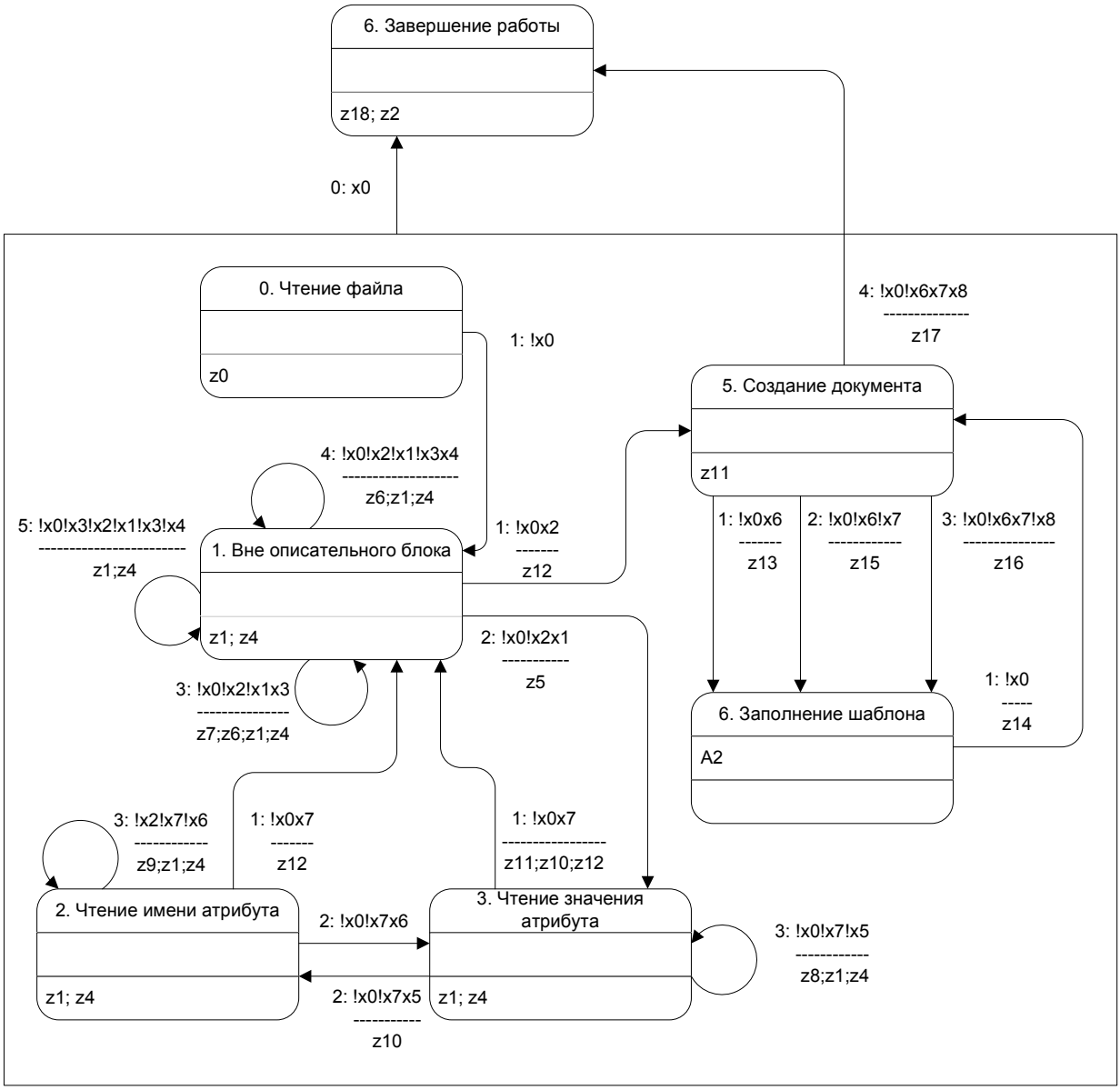

Рис. 11. Граф переходов автомата A1

4.4.3. Автомат заполнения шаблона A2

Формальное описание

Автомат A2 создает файлы документации на основе загруженных шаблонов, он последовательно пробегает шаблон и в нужных позициях, где находятся специальные команды, вставляет полученную автоматом A1 информацию.

Нумерация, перечень и описание состояний автомата A2

В табл. 4 приводится нумерация, названия и описание состояний автомата A2.

Таблица 4. Состояния автомата A2

№	**Название состояния**	**Описание состояния**
0	Чтение шаблона	Загрузка файла шаблона, если он не пришел при инициализации
1	Обработка блоков повтора	Поиск и обработка блоков повтора
2	Размножение блоков повтора	Определение необходимого количества и размножение блоков повтора
3	Заполнение шаблона	Заполнение шаблона для блока повтора
4	Поиск команд условного оператора	Определяет в шаблоне блоки условного оператора и обрабатывает их

5	Обработка команд	Определение команд в шаблоне и замена их свойствами
6	Завершение работы	Завершение работы автомата

Перечень входных переменных автомата A2 и их описание

Ниже дается описание входных переменных автомата A2.

x0 Автомат выдал ошибку

x1 В шаблоне есть необработанный блок

x2 В шаблоне есть необработанный атрибут

x3 Необходимо создать еще одну копию блока повтора

x4 Текст шаблона загружен

x5 В шаблоне есть команда условного оператора

x6 Существует определение атрибуту шаблона

Перечень выходных воздействий автомата A2 и их описание

Ниже перечислены выходные воздействия автомата A2.

z0 Загрузка текста шаблона из файла

z1 Завершение работы автомата A2

z2 Выделение блока повтора из шаблона

z3 Создание автомата для обработки блока повтора

z4 Удаление блока повтора из шаблона

z5 Добавление обработанного блока в шаблон

z6 Замена названия свойства его значением в шаблоне

z7 Сохранение обработанного шаблона

z8 Удаление атрибутов команд начала и конца условного оператора

z9 Удаление содержимого между командами условного оператора из шаблона

Схема связей автомата A2

Схема связей автомата A2 представлена на рис. 12.

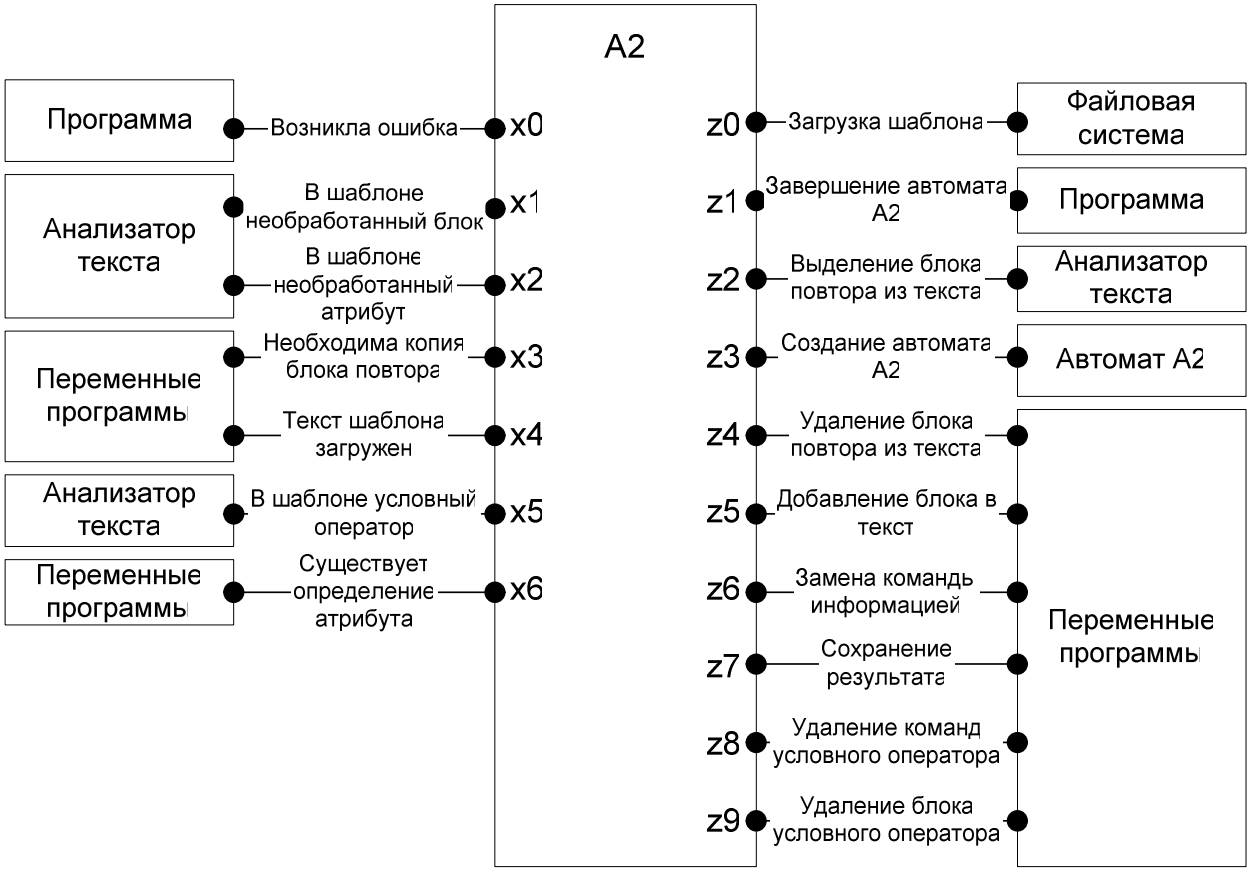

Рис. 12. Схема связей автомата A2

Граф переходов A2

Граф переходов автомата A2 представлен на рис. 13.

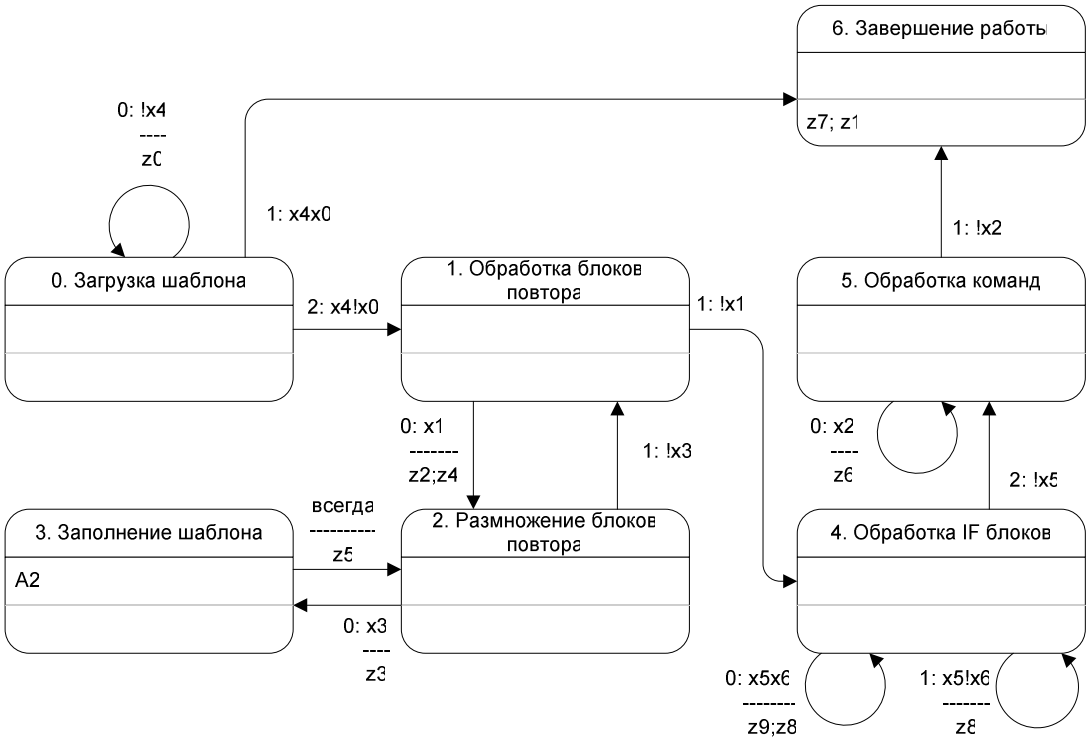

Рис. 13. Граф переходов автомата A2

4.4.4. Автомат обработки правил A3

Формальное описание

Автомат A3 создается для каждого правила каждого блока шаблона. Он осуществляет процесс выделения информации из текста программы. При считывании автоматом A1 очередного символа из текста анализируемого файла формируется событие загрузки символа. Это событие поступает набору автоматов активных правил, которые осуществляют проверку на выполнение правил. В процессе проверки определяется необходимая информация и передается автомату A1.

Нумерация, перечень и описание состояний автомата А3

В автомате А3 предусмотрено три состояния, описанные в табл. 5.

Таблица 5. Состояния автомата А3

№	Название состояния	Описание состояния
0	Начало правила не выполнилось	Обрабатываемое правило не выполнилось
1	Выполнилось начало правила, и не выполнился конец правила	Выполнилось начало обрабатываемого правила
2	Выполнился конец правила	Правило выполнилось полностью, выделен блок информации из текста

Перечень и описание событий автомата А3

Автомат А3 реагирует на два события.

e0 Загружен очередной символ анализируемого текста

e1 Необходимо начать проверку заново

Оба события генерируются в автомате А1. После этого совершается очередная итерация работы автомата А3.

Перечень входных переменных автомата А3 и их описание

Переменные автомата А3 описаны ниже.

x0 Выполнилось начало или конец правила

x1 Выполнилось начало подправила данного правила

Перечень выходных воздействий автомата А3 и их описание

Ниже приведены выходные воздействия автомата А3.

z0 Инициализация проверки начала правила

z1 Сохранение пришедшего символа

z2 Сохранение результата правила

z3 Инициализация проверки конца правила

Схема связей автомата А3

Схема связей автомата А3 представлена на рис. 14.

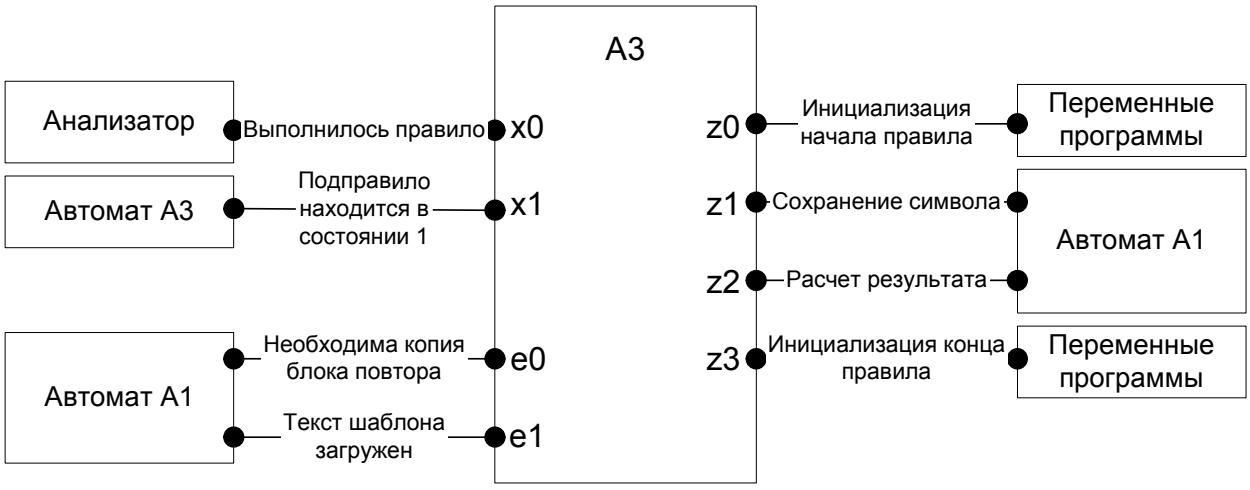

Рис. 14. Схема связей автомата А3

Граф переходов A3

Граф переходов автомата A3 изображен на рис. 15.

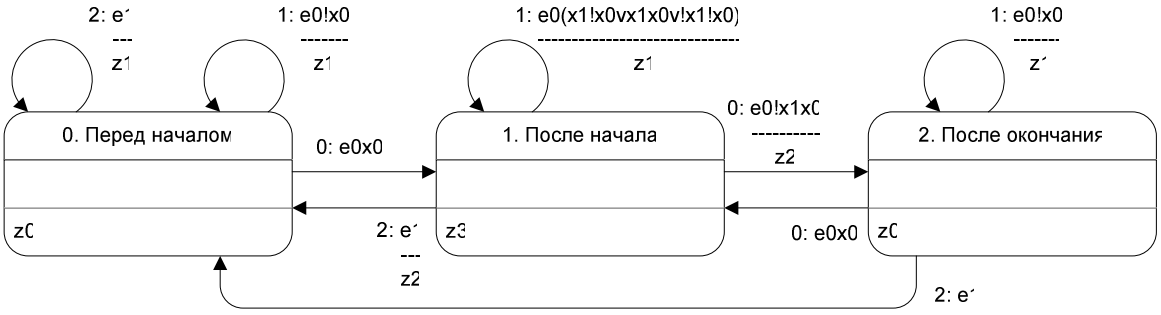

Рис. 15. Граф переходов автомата A3

ГЛАВА 5. МЕТОД НА ОСНОВЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

5.1. Общий принцип

Составной частью системы автоматического документирования, описанной в разд. 4.1, является метод выделения информации из документируемых файлов, основанный на «гибких» правилах. Метод, основанный на теории регулярных выражений [12], является альтернативным. Он был использован в рассматриваемом (втором) варианте системы при определении конструкций правил выделения информации.

Кроме того, изменим методы создания и заполнения шаблонов. Новый метод заполнения шаблонов базируется на метаязыке *XML*. Определяемые в шаблонах заглушки и команды теперь описываются по правилам конструкций-тегов заданного стилевого пространства [13].

По сравнению с методом, базирующимся на «гибких» правилах, рассматриваемый метод включает в себя несколько компонент, каждый из которых выполняет узкую задачу. Схема системы автоматического документирования на основе регулярных выражений изображена на рис. 16.

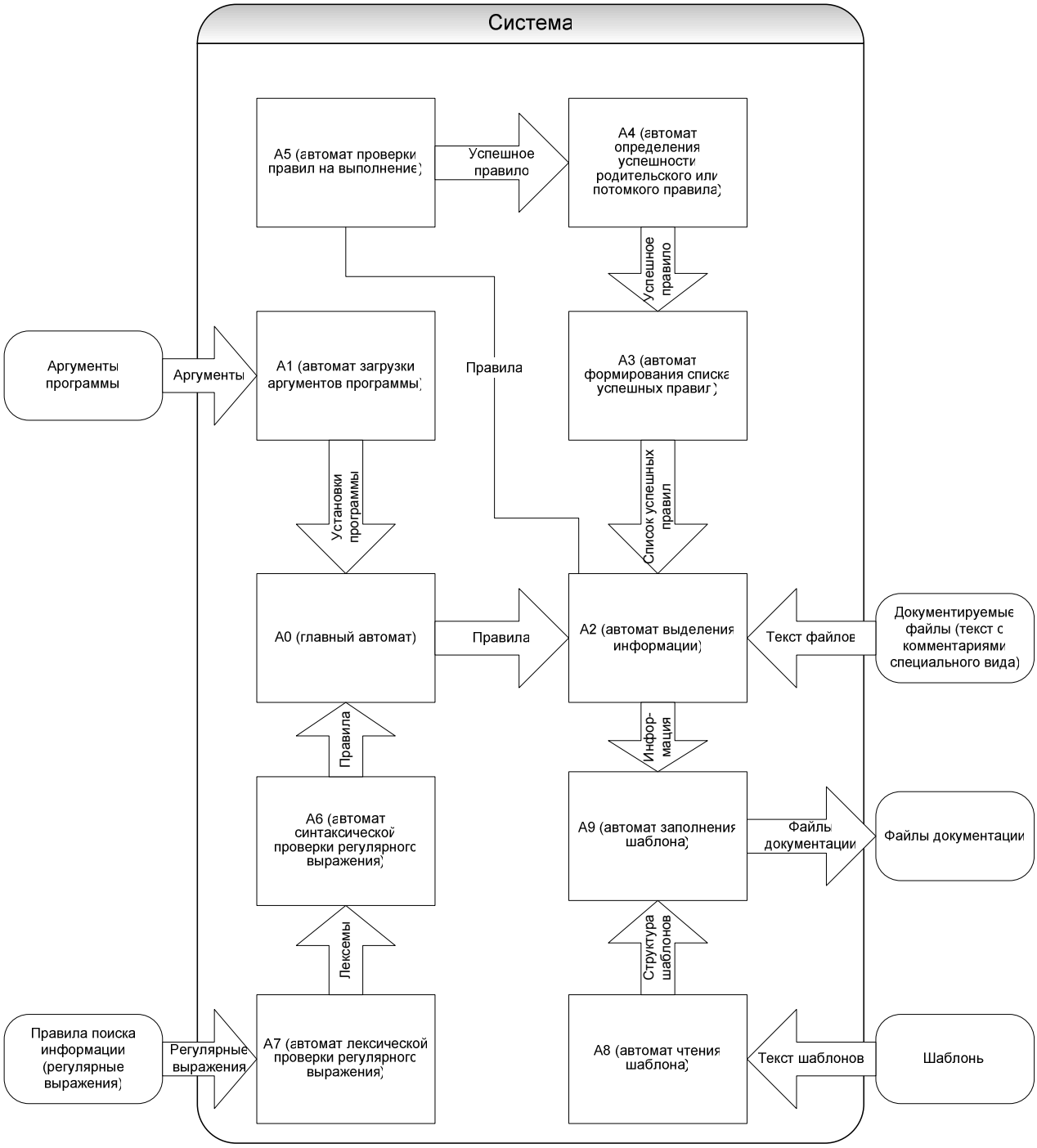

Рис. 16. Система документирования, основанная на регулярных выражениях

Автоматы A0 и A1 отвечают за общее функционирование системы. Автоматы A2 – A7 выделяют фрагменты информации из документируемых файлов, причем автоматы A2 – A5 поэтапно осуществляют задачу выделения информации с помощью правил, считанных и проанализированных

автоматами A6 и A7. Автомат A8 производит загрузку шаблона в систему, а автомат A9 создает на основе этого шаблона файлы документации.

Основной принцип выделения информации, основанный на делении пользователем документируемых файлов на логические или функциональные блоки, остается прежним.

5.2. Регулярные выражения

Регулярные выражения в настоящей работе будут использоваться для описания правил поиска подстроки в строке. Они состоят из последовательности обыкновенных и специальных символов. Специальные символы – это зарезервированные символы (совокупности символов), задающие команды при поиске подстроки.

Примером регулярного выражения может служить слово

```
страницы
```

В этом примере регулярное выражение состоит только из обыкновенных символов. На основе этого выражения можно определить подстроку, состоящую из одного слова, в предложении

```
Это заголовок главной страницы сайта.
```

В рассмотренном примере это слово найдет само себя.

Кроме обыкновенных символов в регулярном выражении может встречаться специальный символ «`.`», который означает «один любой символ». Если он, например, встретится в выражении «`гла.ной`», то оно в рассмотренном примере выделит слово «главной», четвертый символ которого (в данном случае буква «в») мог быть любым.

В регулярном выражении могут использоваться также и суффиксы, которые задают количество повторений символов, стоящих перед суффиксами.

В качестве суффиксов используются следующие символы: «*», «+», «?», «{n,m}». Например, выражение «аб*в» определяет любое слово (в смысле сочетания символов), на концах которого находятся символы «а» и «в», а в середине ноль или более раз повторяется символ «б». Для такого выражения подойдут слова «ав», «абв», «аббв» и т.д. Аналогично, суффикс «+» означает, что символ перед ним может повторяться один или более раз, а «?» - символ перед ним может присутствовать один раз в искомой подстроке или отсутствовать. Суффикс «{n,m}» задает число повторений символов между n и m, где n, m – целые числа (n≤m).

В регулярном выражении может встретиться совокупность символов, обрамленная квадратными скобками «[» и «]». Лишь один из этих символов может встретиться в искомой подстроке. Например, для регулярного выражения «с[ыо]р» подойдет и слово «сор», и слово «сыр». Внутри квадратных скобок может встретиться также диапазон, заданный крайними символами и знаком дефиса «-» между ними. Примером диапазона может быть «[а-яА-Я]», который задает одну букву русского алфавита (строчную и прописную).

В регулярном выражении может встретиться также такое сочетание: «[^a-z]», где ^ – специальный символ. Такое выражение ищет подстроку, в которой не содержится символ из данного множества.

Символ «|» в выражении аналогичен логическому «или». Выражение «абв|где» в этом случае будет соответствовать либо слову «абв», либо слову «где». В сочетании с круглыми скобками «(…)» указанный символ поможет написать сложное выражение для поиска нужной подстроки.

Например, выражение «со(п(ка|ло)|р)» ищет в тексте три слова: «сопка», «сопло» и «сор».

Подвыражение «\x», где x – любой символ в регулярном выражении, добавляет дополнительные возможности и позволяет избежать некоторых конфликтов согласования специальных и обыкновенных символов. Символ «\w» означает в выражении любой буквенный символ, а «\W» – любой не буквенный символ. Аналогично, символ «\d» – любая цифра, а символ «\D» – любой символ, который не является цифрой.

В выражениях обычно не принято применять знаки переноса строки, табуляции, пробела. Вместо них существуют специальные символы (соответственно символы «\n», «\t», «\s»). Примером использования подобных специальных символов может быть выражение для поиска любого слова, обрамленного пробелами, в середине строки: «\s\w+\s».

Символы «\.» или «\(» определяют обыкновенные символы, причем «.» и «(» не будут восприниматься как команды.

Теория регулярных выражений – это развитая теория, которая значительно отличается от ранее введенных «гибких» правил. Выделение фрагмента информации из текста в обоих методах осуществляется на основе двух правил. Первое правило определяет начало выделяемого фрагмента информации, а второе – его окончание.

5.3. Процесс документирования на основе шаблонов

В рассматриваемом методе разработана структура шаблонов на основе *XML* формата. Команды и заглушки шаблонов задаются тегами определенного пространства имен.

Пространство имен – это идентифицируемая с помощью ссылки *URI* коллекция имен, используемая в *XML* документах для обозначения типов

элементов и именования атрибутов [13]. Пространство имен используется внутри тегов по схеме «<[пространство имен]:[имя тега][разделительный символ][атрибуты]>».

Результирующая документация, которая создается на основе шаблонов, не обязана быть набором *XML* документов. Сам же шаблон должен интерпретироваться как *XML*, в котором помимо обычного текста присутствуют командные теги пространства имен разрабатываемой системы. Пространство имен заранее четко задано («*inetDoc*»).

Теги, находящиеся в шаблоне, определяют, каким образом ранее выделенная из документируемых файлов информация вводится в итоговую документацию. Ниже перечислены используемые теги и их назначение.

1. Тэг «<inetDoc:write …>» выводит информацию в заданное место шаблона, если не определен атрибут var. Обязательный атрибут property определяет имя информации, а атрибут name – блок, из которого эта информация выделена. Если задан атрибут var, то вывод информации в шаблон не осуществляется. Вместо этого информация записывается в переменную под именем, определенным в этом атрибуте.

2. Теги «<inetDoc:replace …>» и «<inetDoc:replaceOnce …>» заменяют в информации под именем из атрибутов property и name некоторые символы. Замена производится на основе регулярного выражения, определенного в атрибуте replacewhat. Если подстрока из блока информации соответствует этому выражению, то она заменяется строкой, определенной в атрибуте replaceto. Результат записывается в переменную, заданную атрибутом var. В первом теге, кроме этого, определен атрибут times – максимальное число подстановок, разрешенное этим тегом. Во втором теге это число по умолчанию единица. Атрибут

NOTRETURN позволяет избежать зацикливания при подстановках, осуществляя поиск новой подстановки после уже совершенной.

3. Тег «<inetDoc:save …>» сохраняет информацию, определенную атрибутами property и name, в переменной с именем из атрибута var.

4. Теги «<inetDoc:exist …>» и «<inetDoc:notExist …>» проверяют на наличие и соответственно на отсутствие информации под именем property из блока name. При выполнении условия система просматривает часть шаблона между начальным и конечным тегами. Иначе игнорирует его. Кроме того, в переменную, заданную атрибутом var, записывается результат проверки.

5. Теги «<inetDoc:match …>» и «<inetDoc:notMatch …>» проверяют информацию с именем из атрибута property и name на соответствие регулярному выражению, заданному в атрибуте expression. При соответствии обрабатывается часть шаблона между начальным и конечным тегами. Результат может быть записан в переменную, заданную атрибутом var.

6. Тег «<inetDoc:repeat …>» позволяет вывести часть шаблона между начальным и конечным тегами несколько раз. Число повторений определяется либо атрибутом times, либо количеством блоков информации, определенных одним именем (это имя определяется атрибутами property и name). Во втором случае при очередной итерации в переменной, определенной атрибутом var, будет находиться очередной блок информации. Кроме того, внутри шаблона между тегами начала и конца повторения будет доступна переменная REPEAT_INDEX#, определяющая номер повтора (нумерация начинается с нуля), знак # заменяется числом вложенности тегов повтора.

Для шаблонов могут быть определены некоторые другие теги.

5.4. Автоматная реализация

Рассматриваемый метод автоматического документирования несколько сложнее метода, основанного на «гибких» правилах. В его реализации участвует уже десять автоматов: два из них отвечают за общее функционирование системы, шесть – за загрузку правил выделения информации и собственно ее выделение из документируемых файлов, а два последних – за документирование на основе выделенной информации. Несмотря на обилие автоматов, схема работы системы (рис. 3) остается прежней.

5.4.1. Главный автомат A0

Формальное описание

Автомат A0 управляет программой, формирует список анализируемых файлов и управляет функционированием системы в целом.

Нумерация, перечень и описание состояний автомата A0

Автомат A0 осуществляет переход между состояниями, перечисленными в табл. 6.

Таблица 6. Состояния автомата A0

№	Название состояния	Описание состояния
0	Начало работы	Устанавливаются переменные, переданные командной строкой
1	Загрузка свойств	Загрузка файлов опций и шаблонов
2	Процесс	Определяются список файлов для документирования и автомат-обработчик для

	документирования	каждого из них
3	Обработка файла и заполнение шаблона для него	Запускается автомат A1 для анализа документируемого файла
4	Заполнение шаблона	Запускается автомат A2 для заполнения шаблона по умолчанию для документируемого файла
5	Создание общих файлов	Определяет общие файлы документации
6	Заполнение шаблона для общего файла	Запускает автомат A2 для заполнения шаблона очередного общего файла
7	Завершение работы	Завершение работы программы

Перечень входных переменных автомата A0 и их описание

Ниже перечислены все входные переменные автомата A0.

x0 Массив ошибок не пуст

x1 Автомат-обработчик аргументов завершился с ошибкой

x2 Автоматы для работы с шаблонами завершились с ошибками

x3 Автомат поиска определений завершился с ошибками

x4 Определено достаточно параметров для работы

x5 Существует еще необработанный общий файл

x6 Существует еще необработанный файл

x7 Существует шаблон-обработчик для следующего файла

Перечень выходных воздействий автомата A0 и их описание

Ниже перечислены все выходные воздействия автомата A0.

z0 Завершаем работу автомата и выводим лог

z1 Сохраняем ошибку автомата-обработчика аргументов

z2 Выводим на экран ошибки

z3 Устанавливаем ошибку - недостаточно аргументов для начала работы

z4 Загружаем файлы установок программы и шаблонов

z5 Создаем список файлов для обработки

z6 Удаляем из списка необработанных файлов верхний файл

z7 Удаляем из списка необработанных общих файлов верхний файл

z8 Сохраняем результат поиска информации в файле в хранилище

z9 Записываем результат работы автомата в файл

z10 Записываем результат работы автомата для общего файла в файл

z11 Создаем данные для автомата заполнения шаблона файла с неизвестным расширением

Схема связей автомата A0

Схема связей автомата A0 представлена на рис. 17.

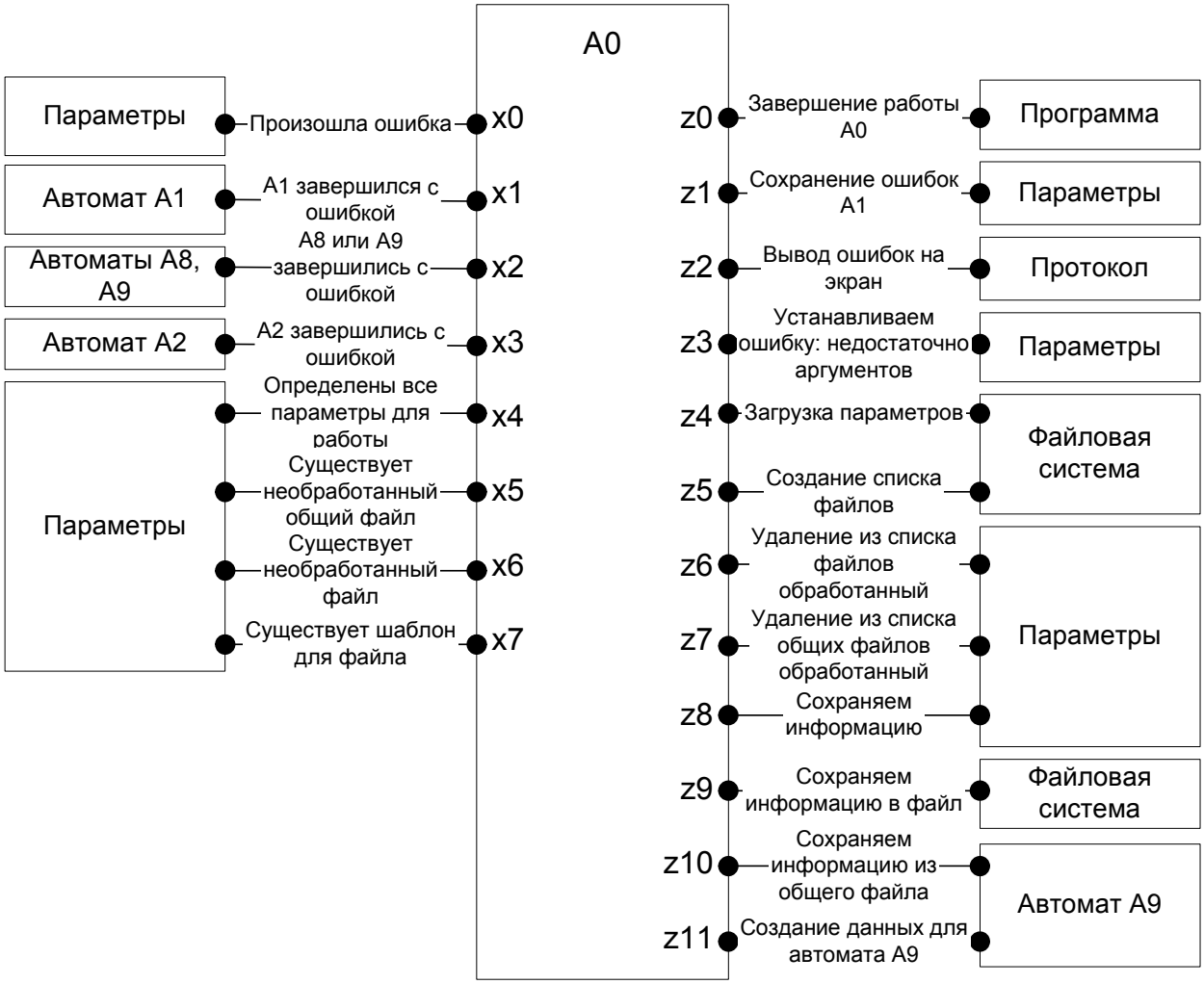

Рис. 17. Схема связей автомата A0

Граф переходов автомата A0

На рис. 18 представлен автомат переходов.

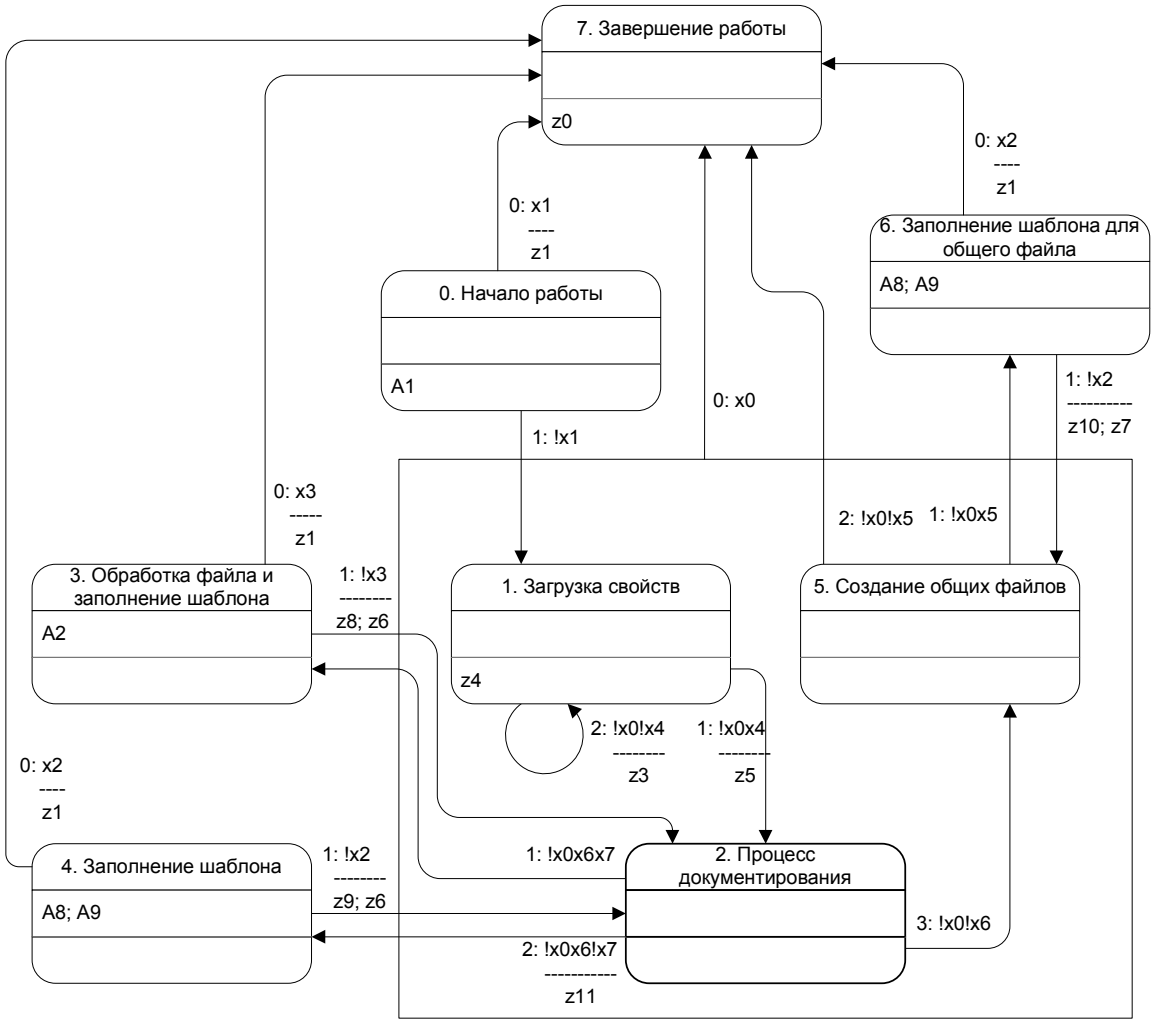

Рис. 18. Граф переходов автомата A0

5.4.2. Автомат загрузки аргументов программы A1

Формальное описание

Автомат A1 занимается загрузкой и обработкой параметров, поступающих от пользователя при запуске системы. Параметры определяют степень протоколирования, файлы, необходимые для документирования, и другие настройки.

Нумерация, перечень и описание состояний автомата A1

Автомат A1 осуществляет переход между состояниями, перечисленными в табл. 7.

Таблица 7. Состояния автомата A1

№	**Название состояния**	**Описание состояния**
0	Начало работы	Вылавливаются ошибки при загрузке параметров, и осуществляется чтение очередного параметра
1	Обработка команды	Обрабатывается команда без параметра
2	Обработка команды с параметром	Обрабатывается команда с параметром

Перечень входных переменных автомата A1 и их описание

Ниже перечислены все входные переменные автомата A1.

x0 Строка аргументов закончилась

x1 Задана команда [-h]

x2 Задана команда [-s]

x3 Задана команда [-l]

x4 Задана команда [-o]

x5 Задана команда [-d]

x6 Задана команда [-p]

x7 Задана команда [-r]

Перечень выходных воздействий автомата A1 и их описание

Ниже перечислены все выходные воздействия автомата A1.

z0 Сохраняем ошибку

z1 Читаем слово как команду

z2 Читаем слово как параметр

z3 Выводим на экран формат запуска файла

z4 Устанавливаем, что программа должна выводить протокол на экран

z5 Устанавливаем, что программа должна выводить протокол в файл

z6 Устанавливаем имя файла протокола

z7 Устанавливаем каталог для готовой документации

z8 Устанавливаем имя файла параметров программы

z9 Устанавливаем имя файла параметров программы

z10 Устанавливаем имя файла для протоколирования успешных правил

z12 Завершаем работу автомата

Схема связей автомата A1

Схема связей автомата A1 представлена на рис. 19.

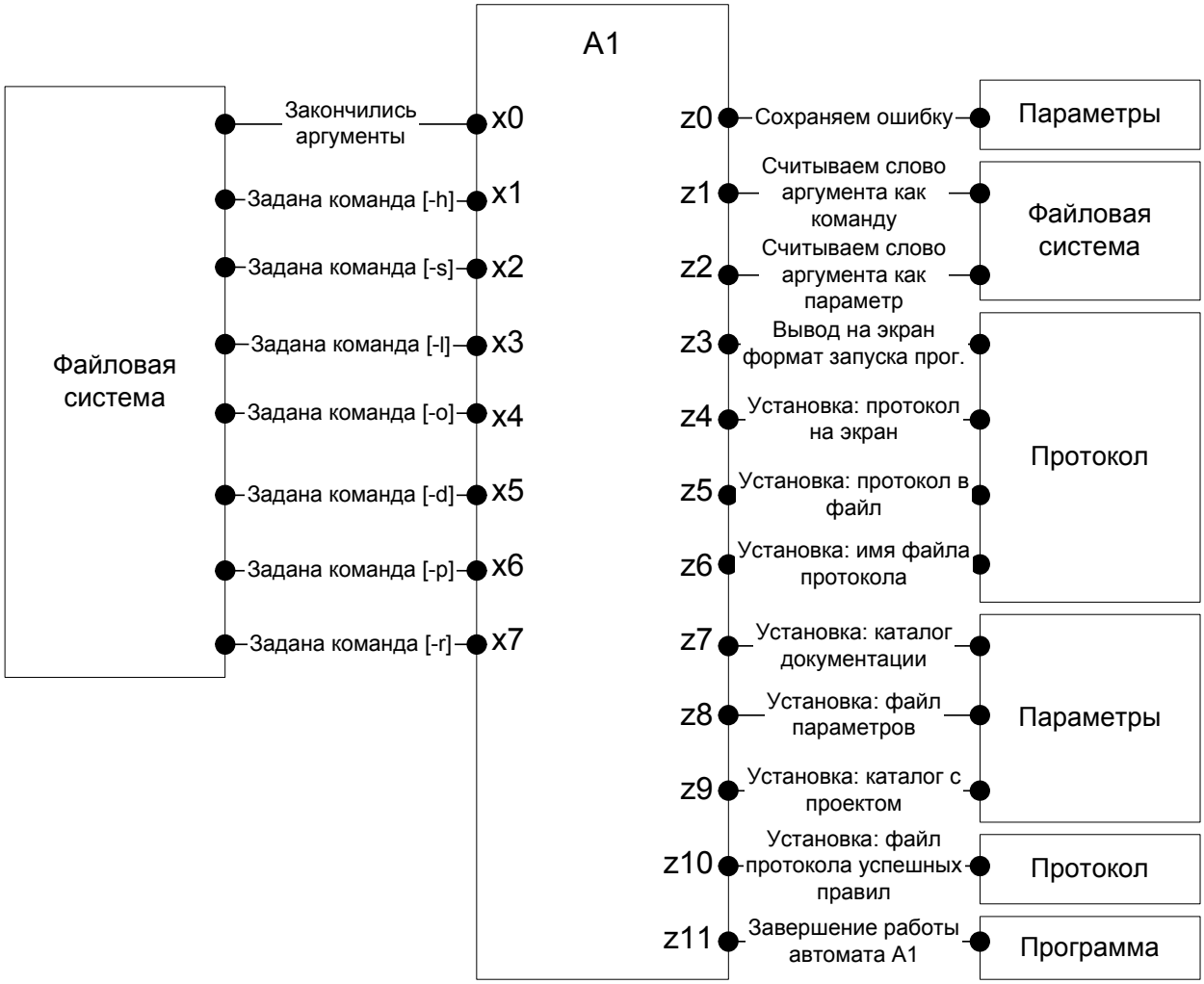

Рис. 19. Схема связей автомата A1

Граф переходов автомата A1

На рис. 20 представлен автомат переходов.

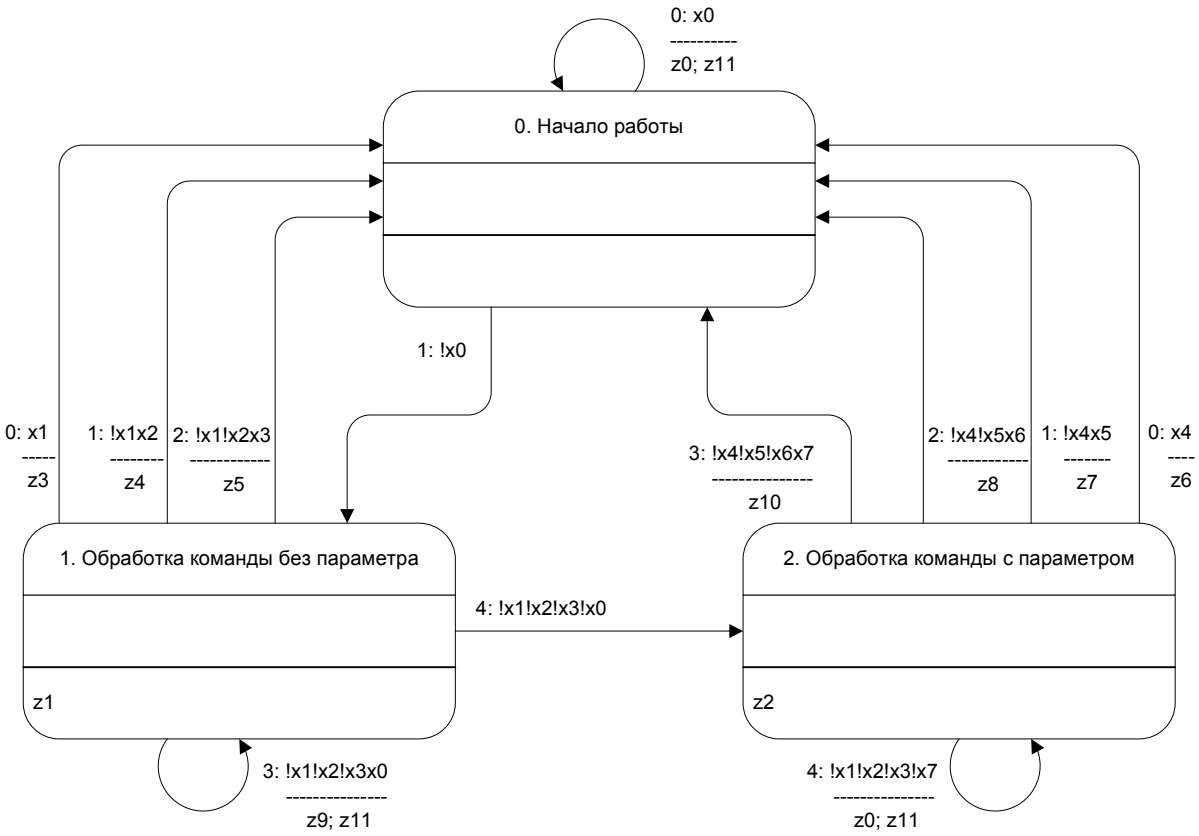

Рис. 20. Граф переходов автомата A1

5.4.3. Автомат выделения информации A2

Формальное описание автомата A2

Автомат A2 анализирует документируемый файл. Он считывает файл, выделяет из этого файла информацию и записывает ее во внутренний формат данных.

Нумерация, перечень и описание состояний автомата A2

В табл. 8 перечислены состояния автомата A2.

Таблица 8. Состояния автомата A2.

№	Название состояния	Описание состояния
0	Чтение файла	Загрузка документируемого файла
1	Проверка главных правил	Проверка правил для переходов между блоками информации
2	Определение сущностных атрибутов	Определение информации, заданной в комментариях специального вида
3	Проверка сущностных правил	Проверка правил выделения информации внутри блоков
4	Заполнение шаблона	Заполнение шаблона
5	Процесс заполнения шаблонов для блоков	Вызов автомата по заполнению шаблонов
6	Заполнение общего шаблона	Заполнение общих шаблонов документации

Перечень входных переменных автомата A2 и их описание

Автомат A2 имеет десять входных переменных, перечисленных ниже.

x0 Массив ошибок не пуст

x1 Файл закончился

x2 Есть необработанные выполнившиеся правила

x3 Очередное выполнившееся правило - завершающее родительское правило

x4 Имя главного правила совпадает с именем одной из сущностей

x5 Завершилось правило, определяющее сущность блока

x6 Текущее главное правило определяет значение атрибута сущности

x7 Текущее сущностное правило определяет значение атрибута сущности

x8 Не для всех блоков заполнены шаблоны

x9 Автоматы выдали ошибки

Перечень выходных воздействий автомата A2 и их описание

Выходные воздействия автомата A2 перечислены ниже.

z0 Чтение файла

z1 Инициализируем проверку правил

z2 Загружаем новый символ

z3 Меняем ссылку главного правила на родителя текущего

z4 Меняем ссылку главного правила на потомка текущего

z5 Удаляем текущее выполнившееся правило из списка

z6 Завершаем все незаконченные правила сущностей

z7 Устанавливаем новый список сущностных правил

z8 Загружаем новый символ для главных правил

z9 Сохраняем результат работы правила

z10 Добавляем начальную строку правилам сущности

z11 Меняем ссылку правила сущности на родителя текущего

z12 Меняем ссылку правила сущности на потомка текущего

z13 Сохраняем результат работы правила

z14 Инициализируем переменные для заполнения шаблонов

z15 Сохраняем ошибки автомата

z16 Сохраняем результат заполнения шаблона

z17 Сохраняем заполненный шаблон

z18 Сохраняем информацию о файле и блоках для использования в шаблонах

z19 Завершаем работу автомата

Схема связей автомата A2

Схема связей автомата A2 представлена на рис. 21.

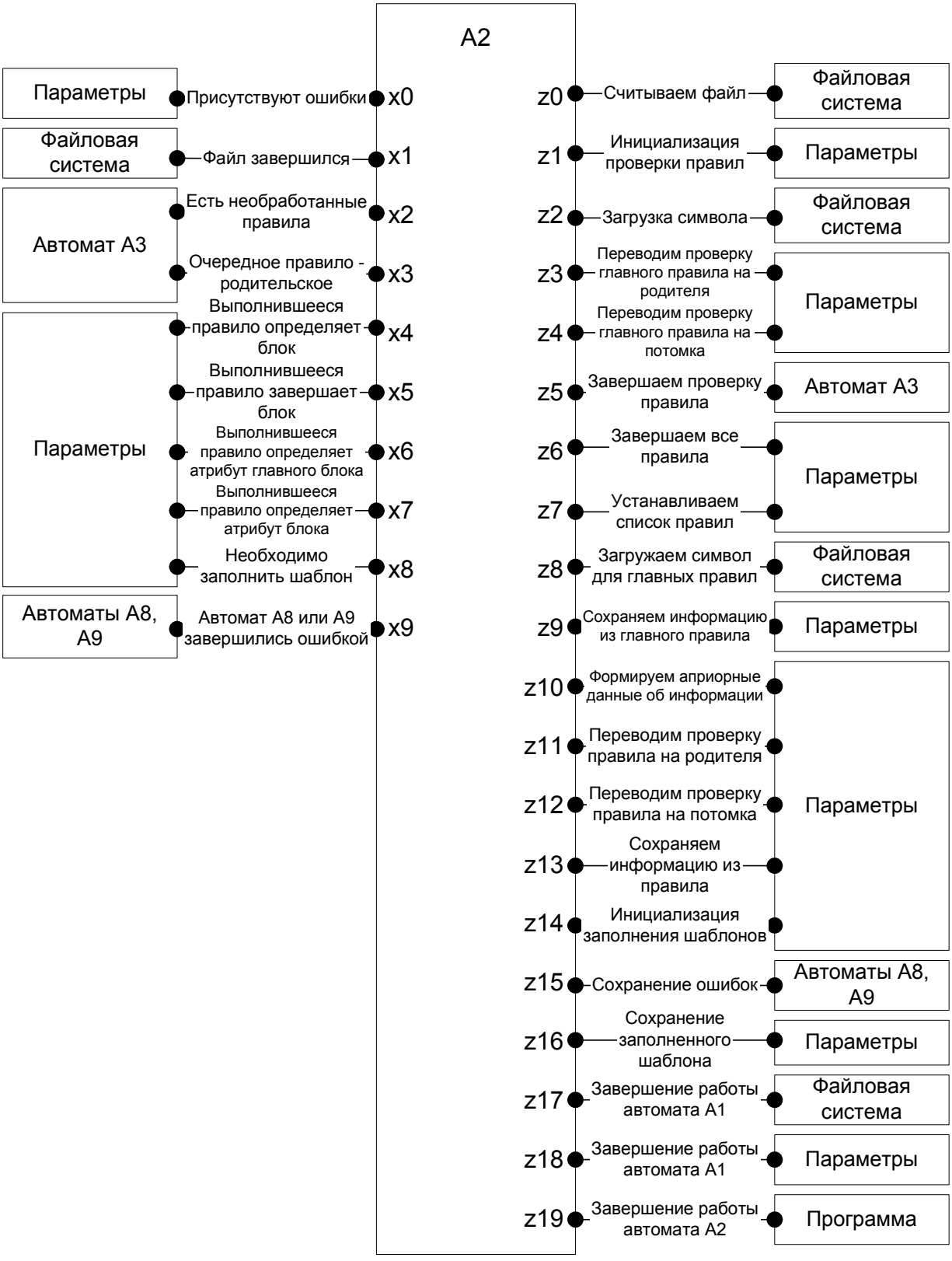

Рис. 21. Схема связей автомата A2

Граф переходов автомата A2

На рис. 22 приведен граф переходов автомата A2.

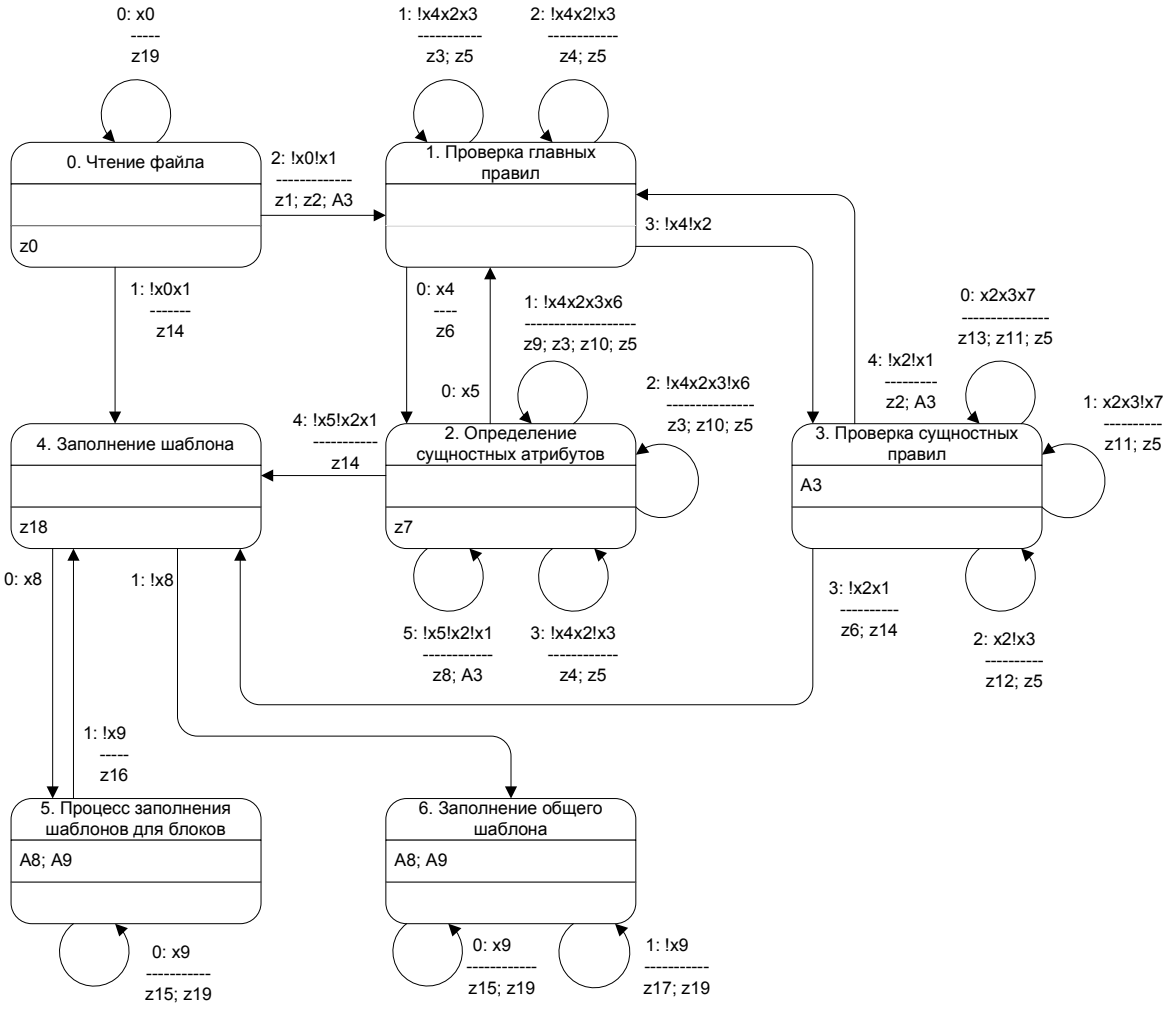

Рис. 22. Граф переходов автомата A2

5.4.4. Автомат формирования списка успешных правил A3

Формальное описание

Автомат A3 формирует список выполнившихся правил выделения информации при считывании из документируемого файла очередного символа. Автомат осуществляет также проверку на зацикливание при переходах между разными уровнями правил.

Нумерация, перечень и описание состояний автомата А3

В табл. 9 перечислены состояния автомата А3.

Таблица 9. Состояния автомата А3.

№	Название состояния	Описание состояния
0	Проверка правил	Проверка всех текущих правил
1	Запись в протокол успешного правила	При успешности какого-либо правила сохранение его в протокол
2	Изменение правила	Формирование нового списка проверяемых правил
3	Проверка на зацикливание	Проверка на зацикливание при переходах в иерархии правил

Перечень входных переменных автомата А3 и их описание

Автомат А3 имеет четыре входных переменных, перечисленных ниже.

x0 Проверка: результат – ни одно правило не выполнилось

x1 Проверка: результат – выполнилось завершающее правило родителя

x2 Произошло зацикливание

x3 Включен режим протоколирования успешных правил

Перечень выходных воздействий автомата А3 и их описание

Выходные воздействия автомата А3 перечислены ниже.

z0 Инициализация переменных автомата

z1 Завершаем работу автомата

z2 Устанавливаем выполнившееся правило как родитель

z3 Устанавливаем выполнившееся правило как потомок

z4 Устанавливаем результат автомата - ошибка

z5 Записываем в протокол программы успешное правило

Схема связей автомата А3

Схема связей автомата А3 представлена на рис. 23.

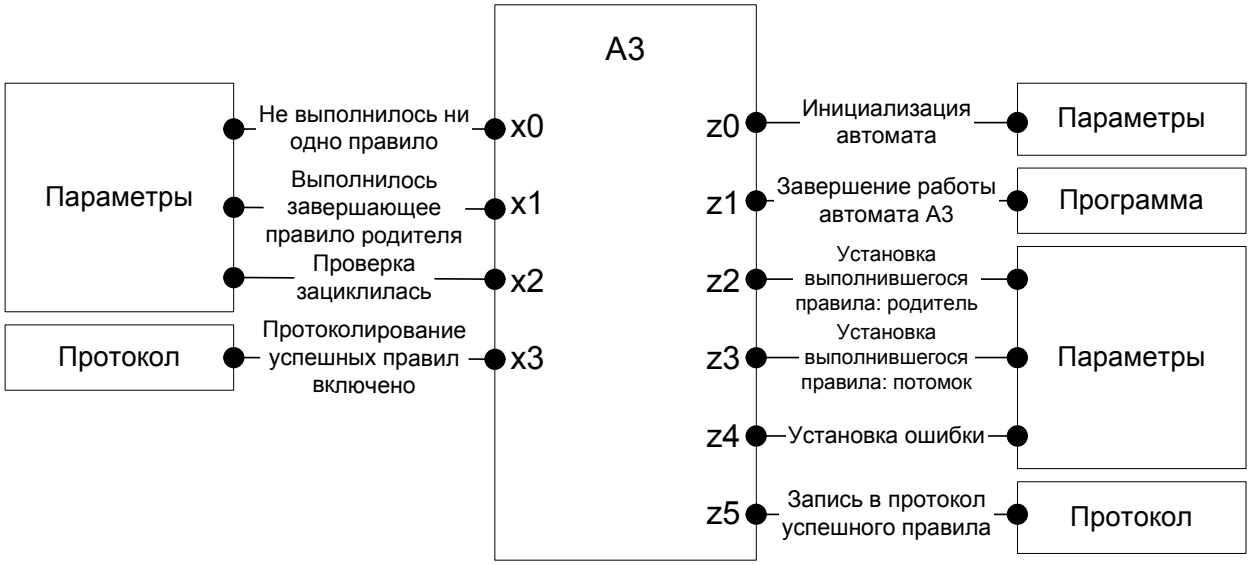

Рис. 23. Схема связей автомата А3

Граф переходов автомата А3

На рис. 24 представлен автомат переходов.

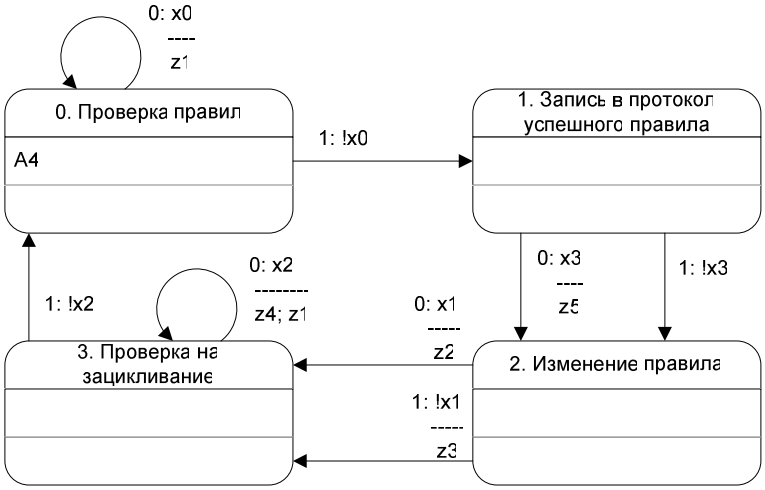

Рис. 24. Граф переходов автомата А3

5.4.5. Автомат определения успешности родительского или правила потомков А4

Формальное описание

Автомат А4 определяет при считывании очередного символа документируемого файла приоритет выполнения правила на окончание текущего блока информации или правила на начало его подблока.

Нумерация, перечень и описание состояний автомата А4

В табл. 10 перечислены состояния автомата А4.

Таблица 10. Состояния автомата А4.

№	Название состояния	Описание состояния
0	Инициализация автомата	Запуск автомата А5
1	Обработка правил потомков	Проверка всех подправил текущего блока информации
2	Обработка правила родителя	Проверка родительского правила
3	Завершение работы автомата	Завершение работы автомата

Перечень входных переменных автомата А4 и их описание

Автомат А4 имеет три входных переменных, перечисленных ниже.

x0 Родительское правило на окончание блока выполнилось

x1 Правило потомка на начало блока выполнилось

x2 Родительское правило более приоритетно

Перечень выходных воздействий автомата А4 и их описание

Выходные воздействия автомата А4 перечислены ниже.

z0 Инициализируем переменные

z1 Проверяем родительское правило на окончание блока

z2 Устанавливаем победу за родительским правилом

z3 Устанавливаем победу за правилом потомка

z4 Устанавливаем, что ни одно правило не выполнилось

z5 Завершаем работу автомата

Схема связей автомата A4

Схема связей автомата A4 представлена на рис. 25.

Рис. 25. Схема связей автомата A4

Граф переходов автомата A4

На рис. 26 представлен автомат переходов.

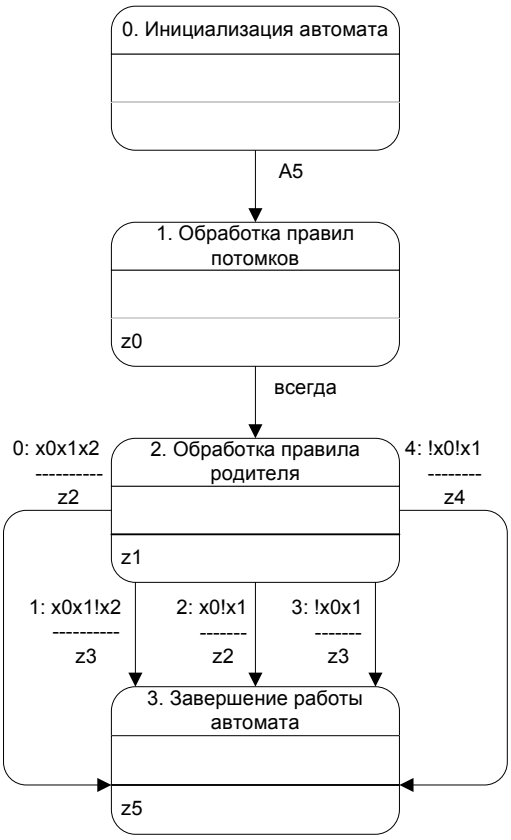

Рис. 26. Граф переходов автомата A4

5.4.6. Автомат проверки правил на выполнение A5

Формальное описание

Автомат A5 при считывании очередного символа документируемого файла проверяет на выполнение одного из подправил текущего блока информации.

Нумерация, перечень и описание состояний автомата A5

В табл. 11 перечислены состояния автомата A5.

Таблица 11. Состояния автомата A5.

№	Название состояния	Описание состояния
0	Инициализация автомата	Инициализация автомата
1	Обработка всех правил	Проверка всех подправил текущего блока информации
2	Выбор результата	Выбор приоритетного успешного правила

Перечень входных переменных автомата A5 и их описание

Автомат A5 имеет три входных переменных, перечисленных ниже.

x0 Все правила были просмотрены

x1 Все результаты проверены

x2 Текущий результат лучше просмотренных результатов

Перечень выходных воздействий автомата A5 и их описание

Выходные воздействия автомата A5 перечислены ниже.

z0 Инициализация массива результатов правил

z1 Обработка очередного правила

z2 Определяем список результатов обработки правил

z3 Сохраняем лучший результат

z4 Удаляем просмотренный результат

z5 Завершаем работу автомата

Схема связей автомата A5

Схема связей автомата A5 представлена на рис. 27.

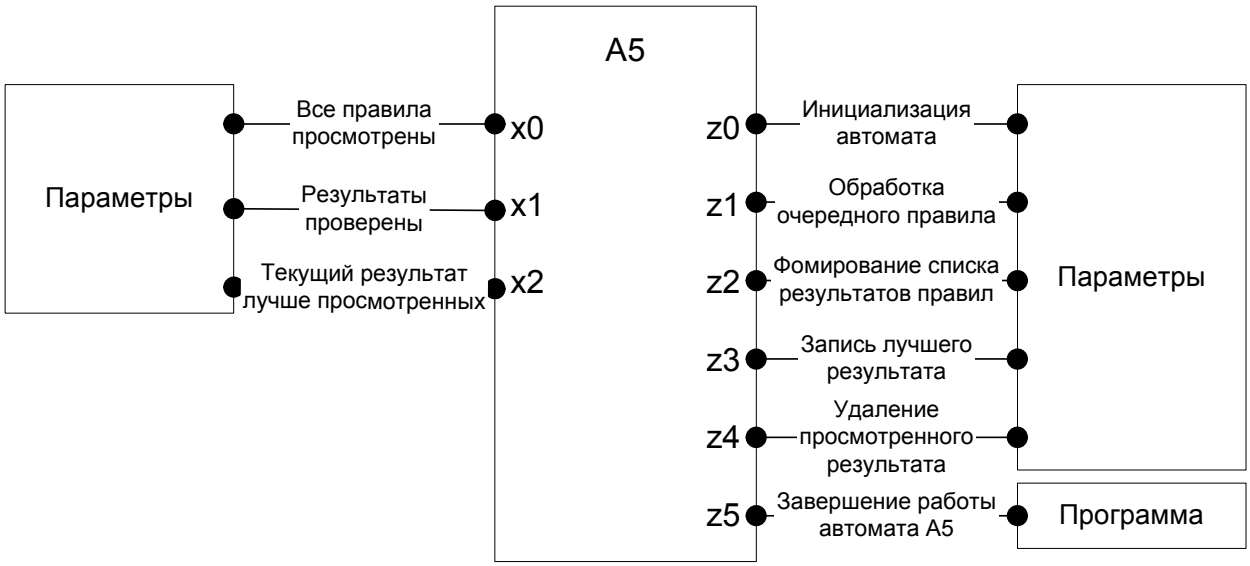

Рис. 27. Схема связей автомата A5

Граф переходов автомата A5

На рис. 28 представлен автомат переходов.

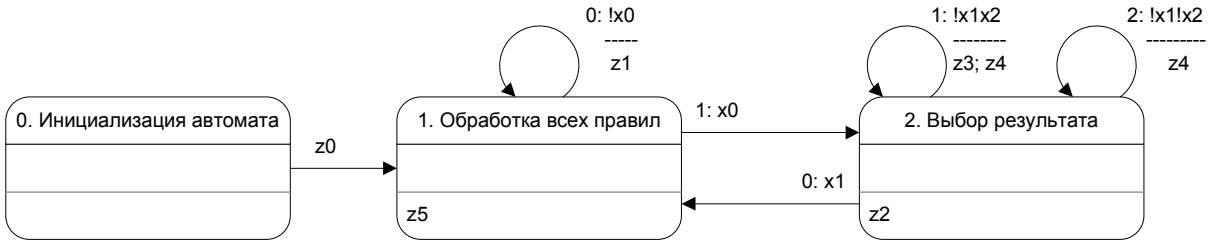

Рис. 28. Граф переходов автомата A5

5.4.7. Автомат синтаксической проверки регулярного выражения A6

Формальное описание

Автомат A6 синтаксически проверяет регулярное выражение на соответствие спецификации.

Нумерация, перечень и описание состояний автомата A6

В табл. 12 перечислены состояния автомата A6.

Таблица 12. Состояния автомата A6.

№	Название состояния	Описание состояния
0	Инициализация автомата	Инициализация автомата
1	Анализ стека	Анализ стека терминалов и нетерминалов
2	Терминал	Обработка терминалов
3	Нетерминал	Обработка нетерминалов
4	Ошибка	Обработка ошибок

Перечень входных переменных автомата A6 и их описание

Автомат A6 имеет семь входных переменных, перечисленных ниже.

x0 Произошла ошибка: лексическая либо другая

x1 На вершине стека команда

x2 На вершине стека терминал

x3 На вершине стека нетерминал

x4 На вершине стека и во входном потоке символ завершения

x5 На вершине стека и во входном потоке терминалы совпадают

x6 Правило вывода существует для данного нетерминала

Перечень выходных воздействий автомата A6 и их описание

Выходные воздействия автомата A6 перечислены ниже.

z0 Инициализируем стек начальным значением S$

z1 Запускаем команду из стека

z2 Вынимаем терминал из стека

z3 Определяем правило вывода

z4 Помещаем правило вывода в стек

z5 Завершение работы автомата

z6 Сохраняем сообщение об ошибке

Схема связей автомата А6

Схема связей автомата А6 представлена на рис. 29.

Рис. 29. Схема связей автомата А6

Граф переходов автомата А6

На рис. 30 представлен автомат переходов.

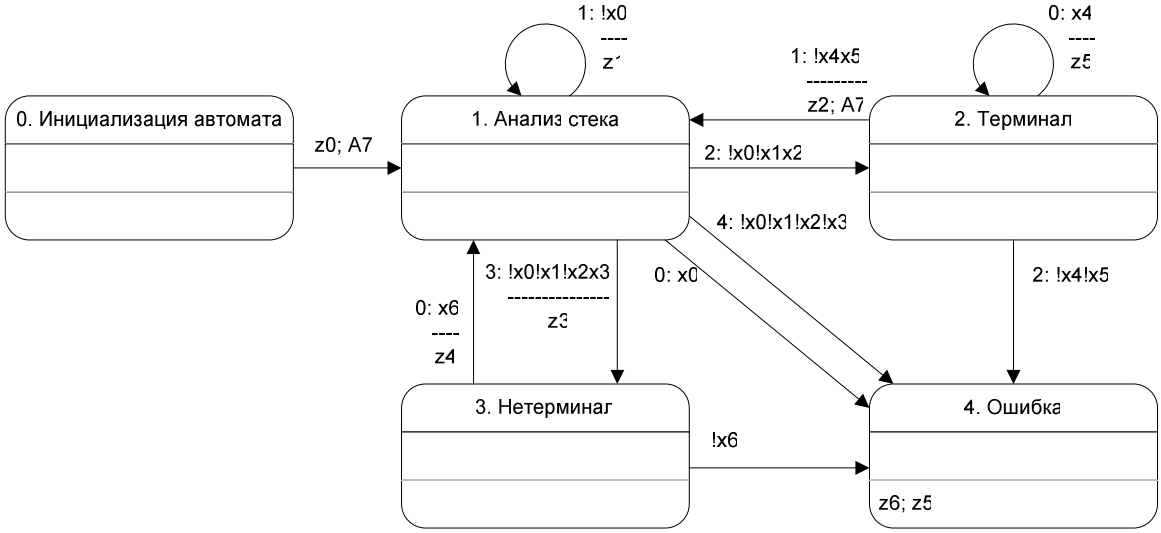

Рис. 30. Граф переходов автомата А6

5.4.8. Автомат лексической проверки регулярного выражения A7

Формальное описание

Автомат A7 лексически проверяет регулярное выражение на соответствие спецификации.

Нумерация, перечень и описание состояний автомата A7

В табл. 13 перечислены состояния автомата A7.

Таблица 13. Состояния автомата A7.

№	Название состояния	Описание состояния
0	Начало работы	Анализ стека терминалов и нетерминалов
1	Лексический анализ	Обработка терминалов
2	Чтение символа	Обработка нетерминалов

Перечень входных переменных автомата A7 и их описание

Автомат A7 имеет пять входных переменных, перечисленных ниже.

x0 Регулярное выражение закончилось

x1 Текущий символ – пробел

x2 Текущий символ - \

x3 Текущий символ – цифра

x4 Текущий символ - один из следующих: (,), ., [,], +, *, ?, |, {, ,, }, -, ^

Перечень выходных воздействий автомата A7 и их описание

Выходные воздействия автомата A7 перечислены ниже.

z0 Завершение работы автомата

z1 Считываем новый символ

z2 Устанавливаем тип лексемы – $- завершение

z3 Устанавливаем тип лексемы – E-ошибка

z4 Устанавливаем тип лексемы - \

z5 Устанавливаем тип лексемы – D

z6 Устанавливаем тип лексемы – один из символов (,), ., [,], +, *, ?, |, {, ,, }, -, ^

z7 Устанавливаем тип лексемы – C-обычный символ

Схема связей автомата A7

Схема связей автомата A7 представлена на рис. 31.

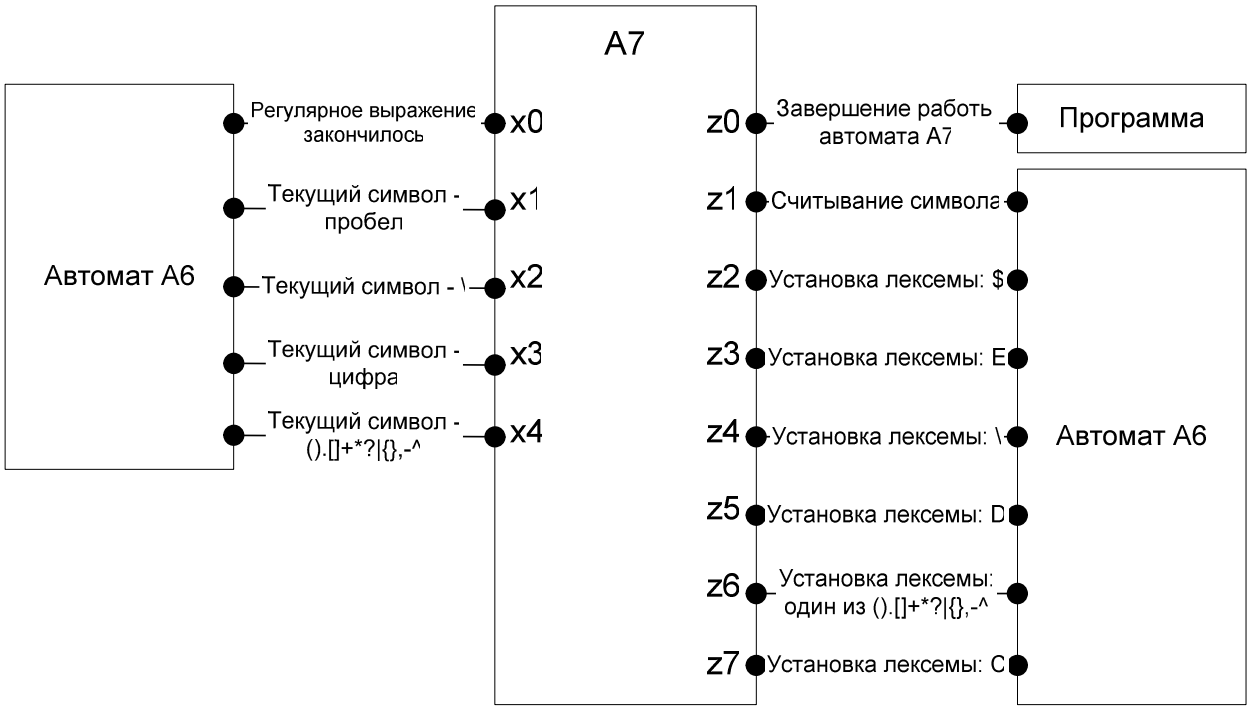

Рис. 31. Схема связей автомата A7

Граф переходов автомата A7

На рис. 32 представлен автомат переходов.

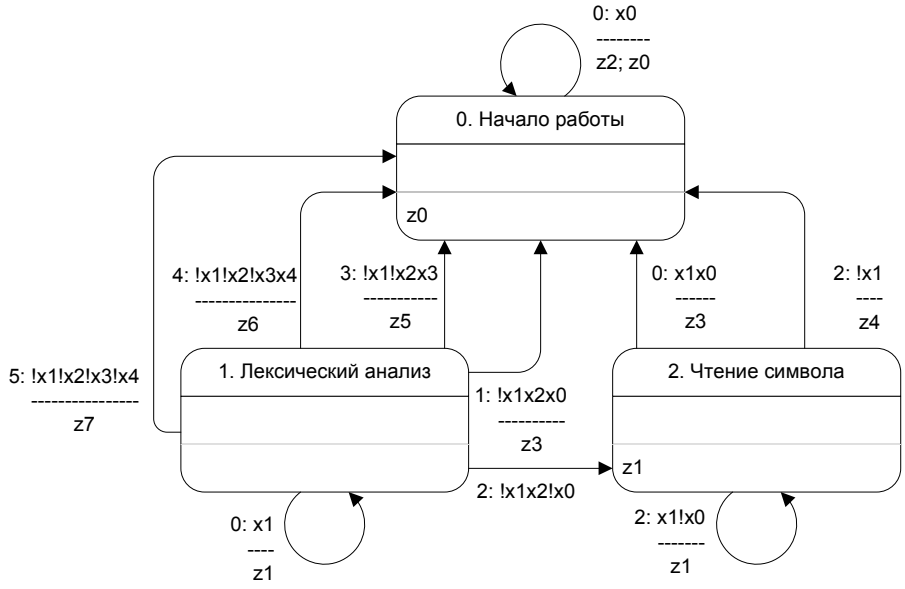

Рис. 32. Граф переходов автомата A7

5.4.9. Автомат чтения шаблона A8

Формальное описание

Автомат A8 считывает файл шаблона и преобразует его во внутренний формат.

Нумерация, перечень и описание состояний автомата A8

В табл. 14 перечислены состояния автомата A8.

Таблица 14. Состояния автомата A8.

№	Название состояния	Описание состояния
0	Инициализация автомата	Инициализация автомата
1	Чтение файла шаблона	Загрузка шаблона
2	Определение узла пространства имен	Выделение узла пространства имен данной системы из текста
3	Обработка узла	Сохранение узла во внутренний формат
4	Обработка атрибутов узла	Сохранение атрибутов узла во внутренний формат

Перечень входных переменных автомата A8 и их описание

Автомат A8 имеет девять входных переменных, перечисленных ниже.

x0 Произошла ошибка

x1 В тексте присутствует необработанный тег данного пространства имен

x2 В тексте необработанному тегу соответствует текст

x3 Очередной тег – открывающий

x4 Имя закрывающего тега совпадает с именем тэга на конце стека

x5 Тег содержит необработанный атрибут

x6 Тег является одиночным

x7 В конце файла нет текстовых элементов

x8 Стек содержит больше одного элемента

Перечень выходных воздействий автомата A8 и их описание

Выходные воздействия автомата A8 перечислены ниже.

z0 Определяем начальные значения переменных

z1 Считываем файл шаблона

z2 Определяем очередной ближайший открывающий или закрывающий тег данного пространства имен

z3 Создаем текстовый элемент документа

z4 Определяем ошибку - синтаксическая ошибка в шаблоне

z5 Закрываем тег, находящийся на верхушке стека

z6 Устанавливаем позицию чтения на следующий символ за символом закрытия тега

z7 Создаем новый элемент документа и кладем его в стек

z8 Сохраняем новый атрибут элемента

z9 Сохраняем последний текст файла

z10 Завершаем работу автомата

Схема связей автомата A8

Схема связей автомата A8 представлена на рис. 33.

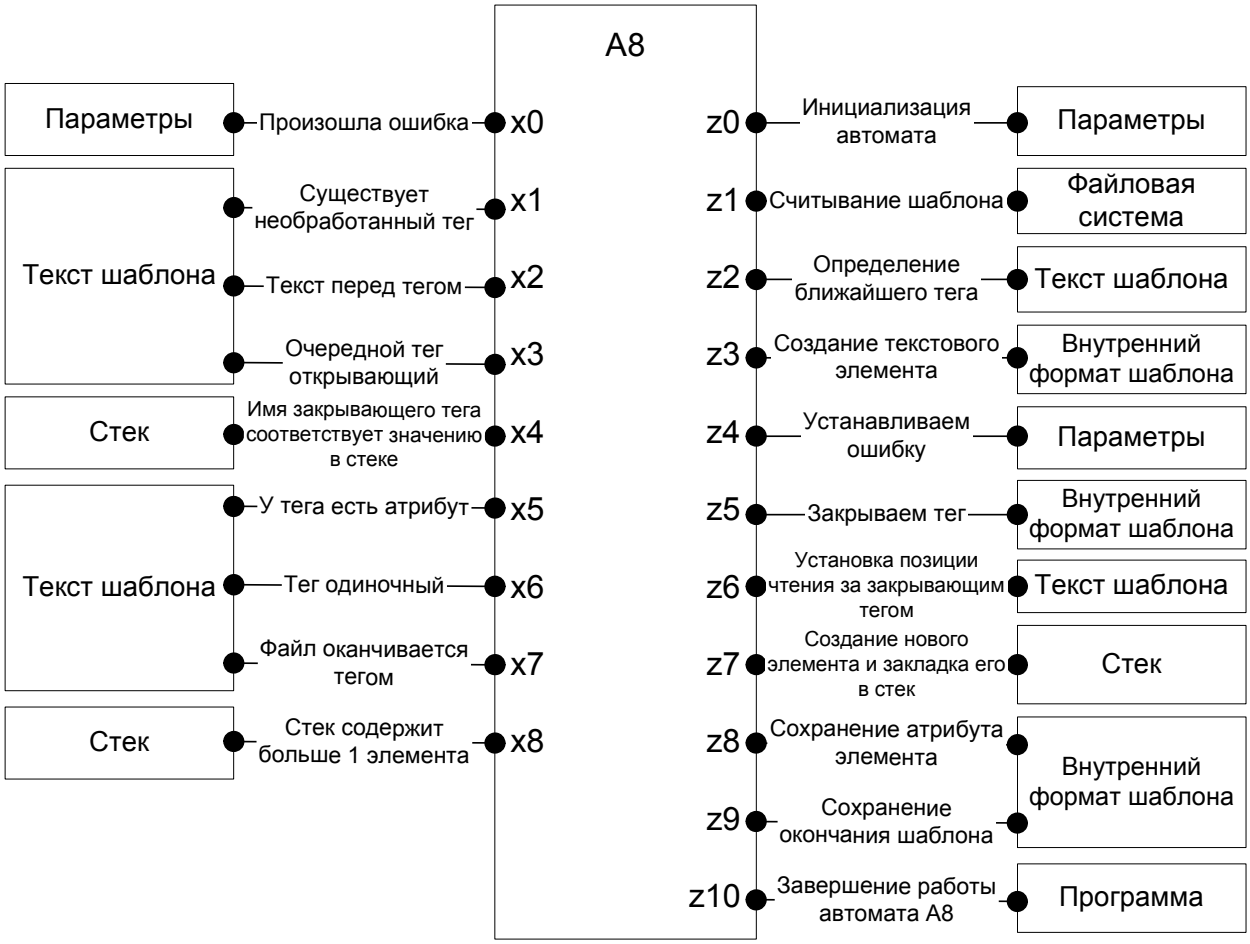

Рис. 33. Схема связей автомата A8

Граф переходов автомата A8

На рис. 34 представлен автомат переходов.

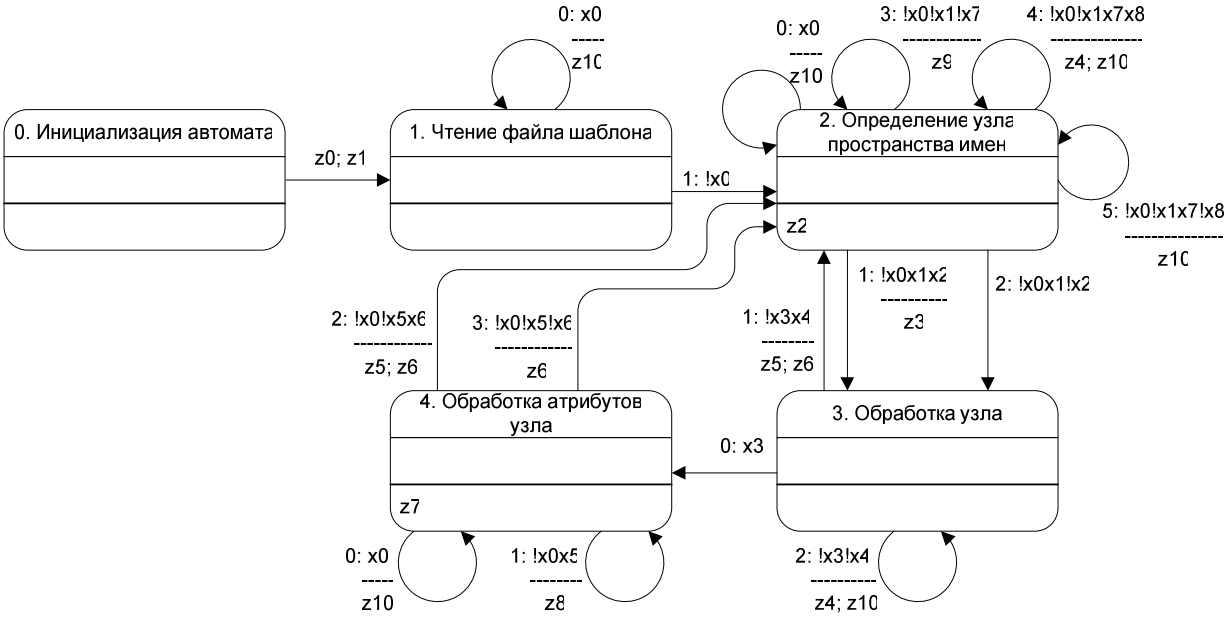

Рис. 34. Граф переходов автомата A8

5.4.10. Автомат заполнения шаблона A9

Формальное описание

Автомат A9 создает файлы документации на основе загруженных шаблонов и выделенной автоматом A2 информации.

Нумерация, перечень и описание состояний автомата A9

В табл. 15 приводится нумерация, названия и описание состояний автомата A9.

Таблица 15. Состояния автомата A9

№	**Название состояния**	**Описание состояния**
0	Инициализация автомата	Инициализация автомата
1	Обработка типа узла	Проверка типа текущего обрабатываемого узла
2	Обработка узлов	Проверка на обработку всех узлов
3	Просмотр узла	Обработка текущего узла
4	Обработка команд	Выполнение команды, задаваемой узлом
5	Обработка узлов-потомков	Обработка узлов-потомков данного узла
6	Заполнение подшаблона	Запуск автомата A9 для подшаблона
7	Сохранение заполненного шаблона	Сохранение заполненного шаблона

Перечень входных переменных автомата A9 и их описание

Ниже дается описание входных переменных автомата A9.

x0 Элемент является атрибутом или текстом

x1 Все элементы-потомки данного элемента обработаны

x2 Элемент-потомок является текстом

x3 Элемент-потомок был обработан с ошибками

x4 Имя элемента-потомка – write

x5 Имя элемента-потомка – replace

x6 Имя элемента-потомка – save

x7 Имя элемента-потомка – repeat

x8 Имя элемента-потомка – exist

x9 Имя элемента-потомка – notExist

x10 Имя элемента-потомка – match

x11 Имя элемента-потомка – notMatch

x12 Имя элемента-потомка – replaceOnce

x13 Имя элемента-потомка – setPage

x14 Имя элемента-потомка – createFile

x15 Имя элемента-потомка – comment

x16 Произошла ошибка

x17 Элемент-потомок содержит другие элементы

x18 Необходимо еще раз прогнать содержимое подшаблона

x19 Необходимо сохранить только что заполненный шаблон

Перечень выходных воздействий автомата A9 и их описание

Ниже перечислены выходные воздействия автомата A9.

z0 Инициализируем работу автомата

z1 Сохраняем значение элемента в результат

z2 Сохраняем значение элемента-потомка в результат

z3 Увеличиваем номер обрабатываемого элемента-потомка

z4 Сохраняем ошибку автомата

z5 Сохраняем результат автомата

z6 Сохраняем ошибку - имя тэга неизвестно

z7 Удаляем обработанный цикл

z8 Обрабатываем тэг write

z9 Обрабатываем тэг repeat

z10 Обрабатываем тэг replace

z11 Обрабатываем тэг replaceOnce

z12 Обрабатываем тэг exist

z13 Обрабатываем тэг notExist

z14 Обрабатываем тэг match

z15 Обрабатываем тэг notMatch

z16 Обрабатываем тэг save

z17 Обрабатываем тэг setPage

z18 Обрабатываем тэг createFile

z19 Сохраняем заполненный шаблон

z20 Завершаем работу автомата

Схема связей автомата A9

Схема связей автомата A9 представлена на рис. 35.

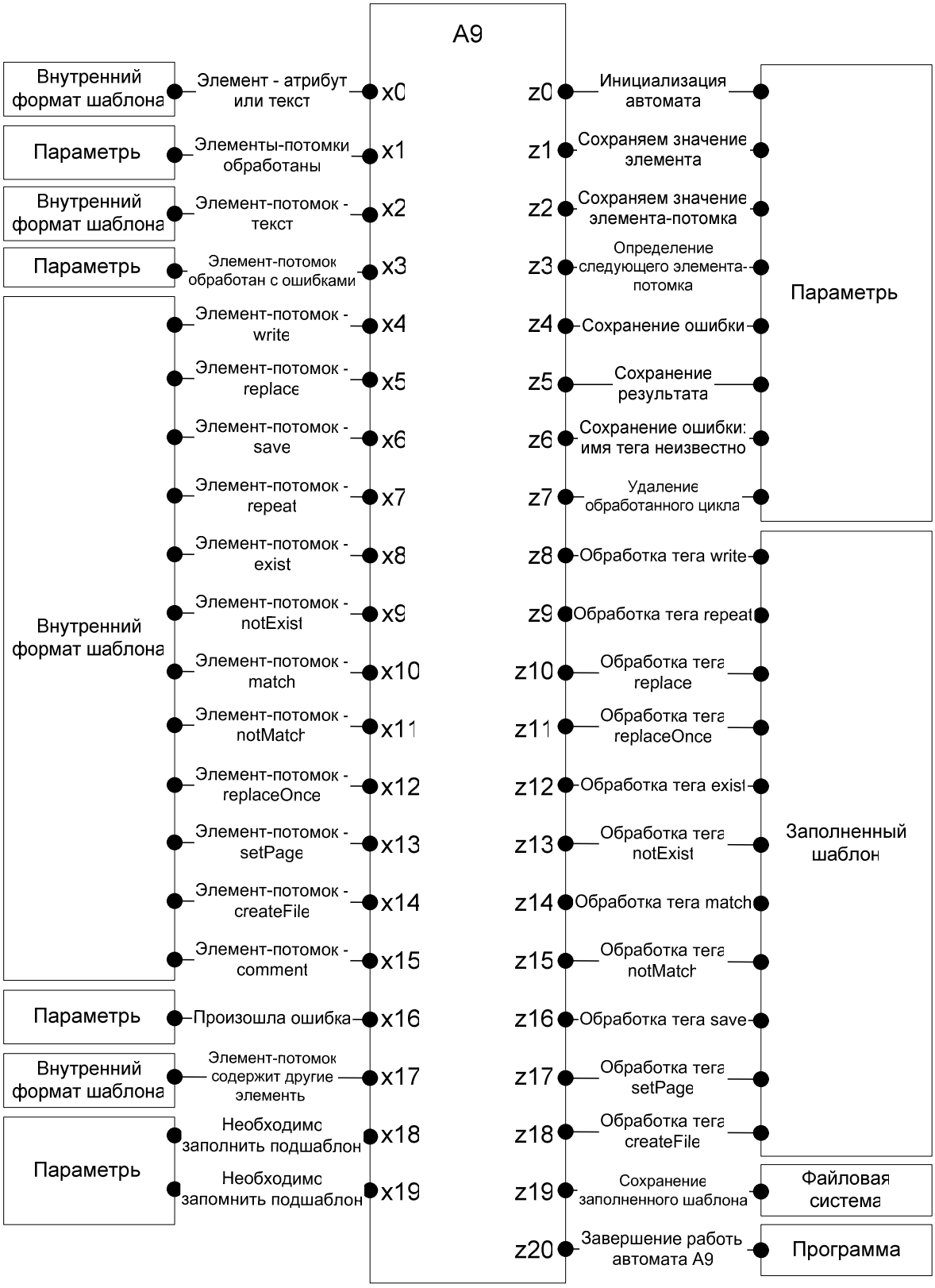

Рис. 35. Схема связей автомата A9

Граф переходов A9

Граф переходов автомата A9 представлен на рис. 36.

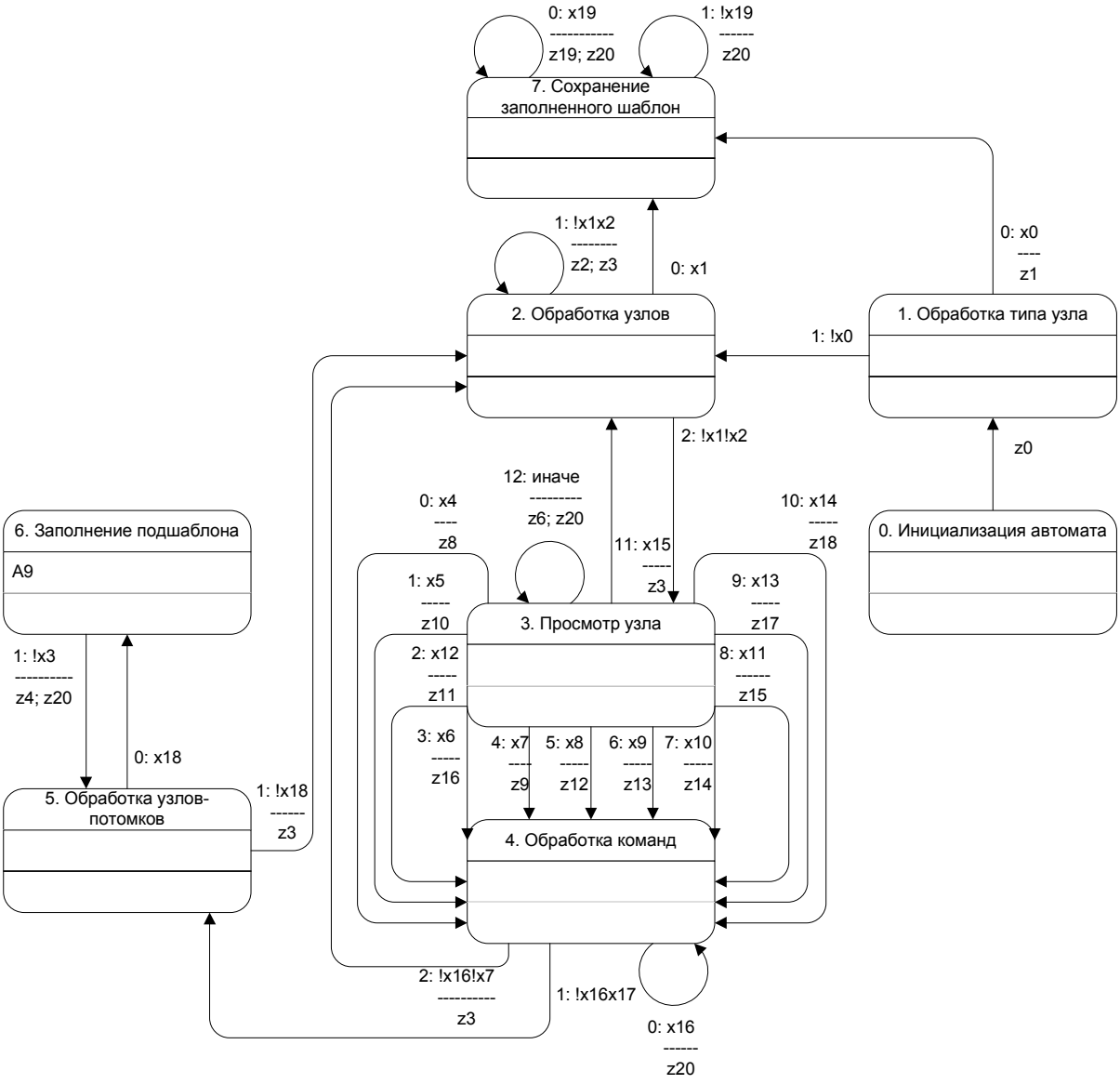

Рис. 36. Граф переходов автомата A9

ГЛАВА 6. МЕТОД С ИСПОЛЬЗОВАНИЕМ РАЗБОРА РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ НА ОСНОВЕ КЛАССИЧЕСКОГО ПОДХОДА

6.1. Общий принцип работы

Предложенный в предыдущей главе метод построения автоматов по регулярным выражениям отличается от построения классических автоматов, построение которых описано в книге [14]. Ниже для общности рассмотрено построение автоматов, базирующееся на классическом подходе.

В книге [14] показано, что регулярные выражения аналогичны правилам переходов между состояниями. В рассматриваемом методе регулярные выражения, которые описывают правила поиска фрагментов информации в тексте, преобразуются в автоматы.

Автоматы, описанные в этой книге, задаются отличным от способа, описанного в работе [3]. Они четко не формализованы, и в них, например, отсутствуют такие понятия, как связь между автоматами, выходные воздействия при переходе между различными состояниями. Кроме того, рассматриваемые в книге автоматы могут быть недетерминированными.

В книге описывается механизм перевода недетерминированных автоматов в детерминированные, но он не будет использован в рассматриваемом методе. Это связано с экспоненциально возрастающим при его применении числом состояний автомата. В рассматриваемом методе автоматы, описанные на основе [14], будут храниться в недетерминированном виде.

Рассматриваемый метод – это модификация второго метода автоматического документирования (глава 5). Структура автоматов в нем осталась прежней, но в качестве внутреннего формата регулярных выражений используются автоматы из книги [14].

6.2. Предлагаемый метод формирования классических автоматов

Создание классических автоматов [14] в системе происходит в несколько этапов. Лексический анализатор загружает регулярные выражения и формирует на выходе лексемы [10]. Они поступают синтаксическому анализатору, который подает команды на вход построителю деревьев, описанному ниже. Деревья строятся и передаются генератору автоматов, который создает на их основе классические автоматы. В дальнейшем эти автоматы используются системой для выделения информации из документируемых файлов.

В настоящей работе, в качестве промежуточного формата хранения данных предлагается использовать деревья [15]. Каждый лист дерева хранит обычный или специальный символ регулярного выражения. Листья разнесены по уровням, структура которых определяется скобочной последовательностью внутри регулярного выражения. Если в выражении встречается подвыражение в скобках, то в дереве на его месте формируется ячейка, потомками которой будут элементы внутри скобок. Потомки каждой ячейки дерева расположены в порядке, в котором они встречаются в регулярном выражении. Суффиксы к символам регулярного выражения «`*`», «+», «`?`» и «`{n,m}`» находятся вместе с этими символами в тех же ячейках.

Пример преобразования регулярного выражения в дерево описанной структуры приведен на рис. 37.

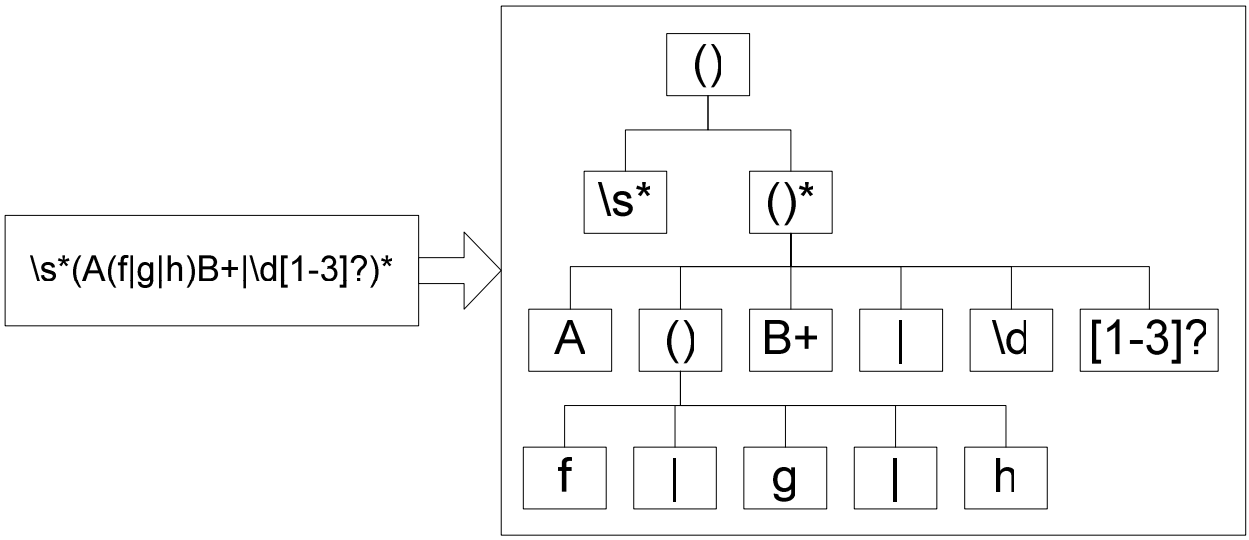

Рис. 37. Пример преобразования регулярного выражения в дерево

Сформированное дерево регулярного выражения поступает генератору автоматов. Он формирует классический автомат, у которого условия переходов между состояниями заданы в листьях дерева. Пример автомата для приведенного дерева регулярного выражения (рис. 37) представлен на рис.38.

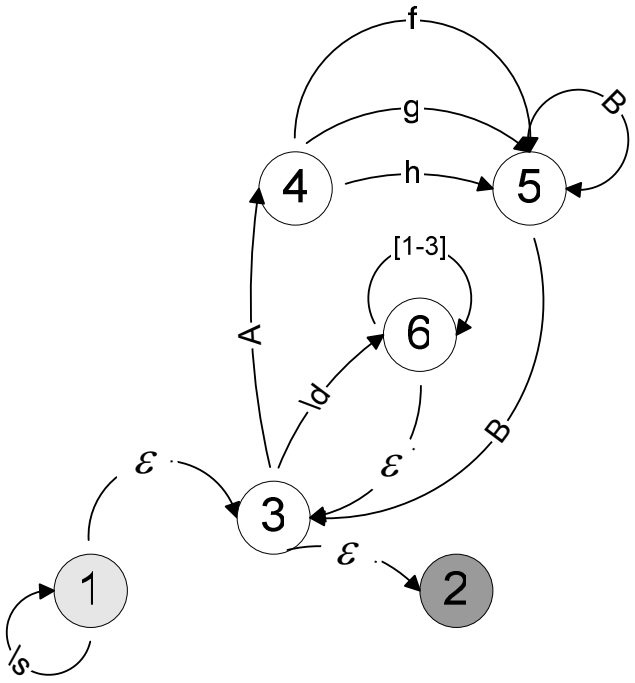

Рис. 38. Пример графа переходов классического автомата

На основе графа переходов классического автомата может быть осуществлен поиск информации в тексте. При поиске идет поочередное сравнение символов текста с условиями на переходах между состояниями графа. При пустом условии переход происходит всегда. Начало прохода по графу осуществляется из состояния 1 и всегда завершается состоянием 2.

Классический автомат в общем случае является недетерминированным. Поэтому из одного состояния по заданному условию может существовать несколько путей перехода в другие состояния. Во время прохода между состояниями учитываются все возможные варианты переходов. Это достигается с помощью массива состояний. Во время очередной итерации цикла проверка условий на переходах происходит для всех состояний, в которые был осуществлен переход на предыдущих итерациях.

Построение классического автомата происходит на основе шаблонов автомата для каждого элемента регулярного выражения. Эти элементы отсортированы по узлам и листьям дерева. Для создания автомата дерево обходится, а к каждому его элементу применяется соответствующий шаблон.

Построение автомата происходит рекурсивно. На обратном шаге рекурсии шаблоны автоматов объединяются в один автомат.

В каждом шаблоне определено начальное и конечное состояния. Они используются в процессе объединения.

Автором настоящей работы выделено семь шаблонов для различных вариантов элементов регулярных выражений.

Последовательность символов в регулярном выражении преобразуется в часть автомата с тремя состояниями. При выполнении первого условия автомат переходит во второе состояние. После этого происходит проверка второго условия. При успехе второй проверки автомат переходит в

завершающее третье состояние. Схема преобразования последовательности символов во фрагмент классического автомата представлена на рис. 39.

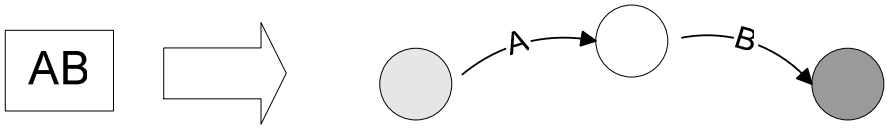

Рис. 39. Схема преобразования последовательности символов регулярного выражения во фрагмент классического автомата

В регулярном выражении может встретиться символ «|», аналогичный логическому операнду «ИЛИ». В этом случае применяется шаблон автомата, в котором между начальным и конечным состояниями существует несколько связей, количество которых зависит от числа «операндов» между описанными символами. Шаблон для данного случая представлен на рис. 40.

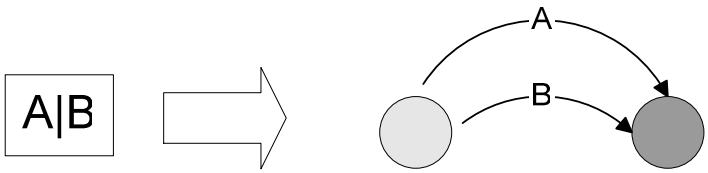

Рис. 40. Схема преобразования подвыражения с символом «|» регулярного выражения во фрагмент автомата классического

Скобки внутри регулярного выражения выделяют подвыражение. Они преобразовываются во фрагмент классического автомата, как показано на рис. 41.

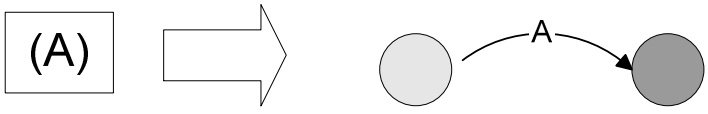

Рис. 41. Схема преобразования подвыражения со скобками регулярного выражения во фрагмент классического автомата

Кроме основных типов конструкций в регулярном выражении также встречаются суффиксы к символам или специальным символам.

Суффикс «*» с символом перед ним преобразовывается в автомат, как показано на рис. 42. Как видно из рисунка, вставляется пустой переход между начальным и конечным состояниями. Этот переход позволяет избежать коллизий во время слияния разных шаблонов в итоговый автомат.

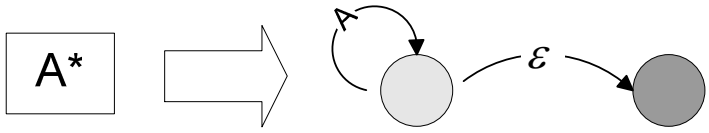

Рис. 42. Схема преобразования подвыражения с суффиксом «*» регулярного выражения во фрагмент классического автомата

Для суффиксов «+» и «?» преобразования представлены на рис. 43, 44.

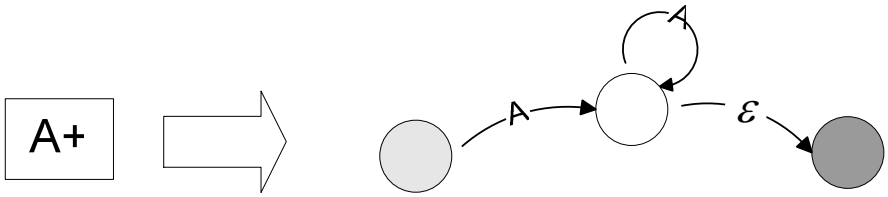

Рис. 43. Схема преобразования подвыражения с суффиксом «+» регулярного выражения во фрагмент классического автомата

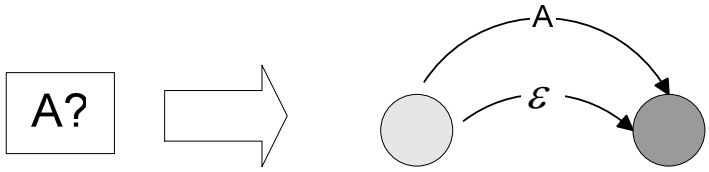

Рис. 44. Схема преобразования подвыражения с суффиксом «?» регулярного выражения во фрагмент классического автомата

Сложнее преобразование осуществляется для случая суффикса «{n,m}» (рис. 45).

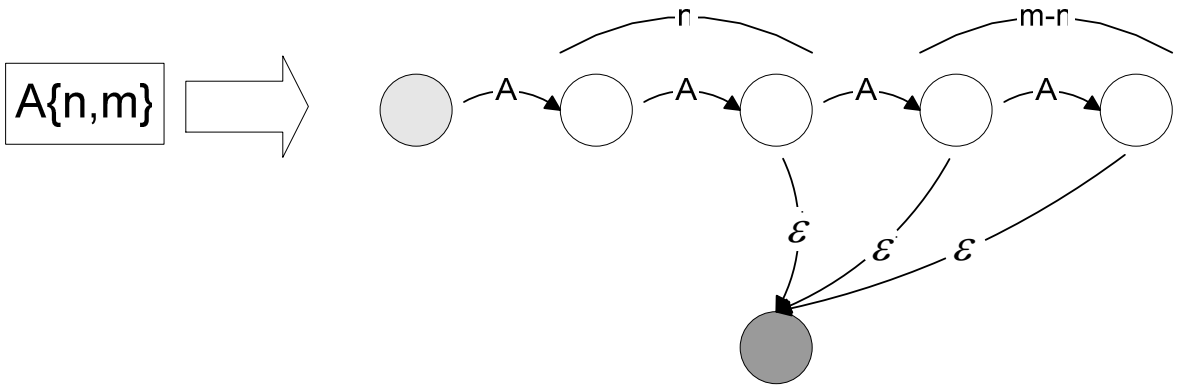

Рис. 45. Схема преобразования подвыражения с суффиксом «{n,m}» регулярного выражения во фрагмент классического автомата

В результате описанных преобразований создается автомат, подобный автомату, представленному на рис. 38. В дальнейшем этот автомат используется при поиске информации в тексте.

Оставшиеся элементы третьего метода автоматического документирования аналогичны второму методу.

ГЛАВА 7. ПРИМЕР АВТОМАТИЧЕСКОГО ДОКУМЕНТИРОВАНИЯ. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Для автоматического создания документации программного проекта необходимо не только наличие исходного текста проекта, на основе которого она будет создаваться. Также обязательными для каждого формата файла проекта являются шаблоны документации и файлы настроек с правилами выделения информации из документируемых файлов.

Ниже описываются некоторые настройки, необходимые для процесса автоматического документирования *Java*-проектов.

7.1. Документирование Java-проектов

Проект на языке программирования *Java* состоит из файлов с расширением «`.java`». Каждый файл описывает языковую конструкцию – класс. Каждый класс характеризуется пакетом – каталогом в структуре проекта. Классы в разных каталогах могут иметь одинаковые имена. В составе классов могут находиться такие элементы языка, как поля и методы.

В качестве документации удобно иметь список классов, отсортированный по пакетам (каталогам). Для каждого класса должны быть описаны поля и методы с их параметрами. В документации также могут присутствовать комментарии программистов к классам, полям и методам.

Документация должна генерироваться на основе исходного кода классов, комментариев, прописанных программистами в этих классах, а также файлов настроек и шаблонов документации.

Каждый класс проекта должен быть разбит на блоки. Для языка программирования *Java* блоки могут выделять конструкции языка: описание класса, метода, поля. Таким образом, программа автоматического документирования будет отдельно обрабатывать каждый блок и действовать в соответствии с типом блока.

Рассмотрим один из блоков исходного текста (полный текст исходного кода класса со всеми выделенными блоками приведен в приложении 1):

```
/*@java
 @Created by Dmitriy I. Suyasov
 @Date 05.04.2005
 @Time 17:00:50
 */
package dsuyasov.inetDoc.automat;

import dsuyasov.inetDoc.functionality.Log;

public abstract class Automat {
…
}
```

Вначале блока располагается комментарий, в котором определяется тип блока – «`java`». Кроме того, в этом комментарии программистом задается имя разработчика и время создания класса. Эту информацию нельзя выделить из исходного текста.

Другие блоки могут иметь другие типы и обрабатываться иначе, чем основной блок.

Ниже рассматриваются основные этапы автоматического создания документации для указанного блока.

7.2. Правила извлечения информации

Перед процессом создания файлов документации системе необходимо извлечь необходимую информацию из исходных файлов *Java*-проекта.

Извлечение происходит на основе правил. Они могут быть либо «гибкими» (разд. 4.2), либо основанными на регулярных выражениях (разд. 5.2).

Так как каждый документируемый файл разделен на блоки, то для выделения информации необходимо сначала найти требуемый блок, а затем определить в нем границы искомого фрагмента информации. Поэтому используется два набора правил выделения информации, первый из них определяет границы блоков, а второй – границы информации в блоках.

Например, для определения начала основного блока в его исходном тексте необходимо найти сочетание символов «`/*@java `». Для его поиска необходимо использовать «гибкие» правила, которые могут выглядеть следующим образом:

```
!/*|!@|!java\n
```

или правила на основе регулярных выражений –

```
/\*@java[\s\n\t\r]
```

Внутри основного блока необходимо выделить несколько фрагментов информации. В частности, в документации понадобится информация об имени класса. Выделение имени может происходить в терминах «гибких» правил:

```
!class,interface|*\s,\n,\t,\r|!ex(\s,\n,\t,\r) и !\s,\n,\t,\r
```

а также на основе регулярных выражений

```
(class|interface)[\n\t\r\s]+\w и [\n\t\r\s{].
```

И в том, и в другом случае из текста будет выделено название «`Automat`».

Полный набор правил для документирования *Java*-файлов приведен в приложениях 2 и 3.

7.3. Шаблоны документации

В исходном тексте *Java*-файла для каждого выделенного блока заполняется свой шаблон – фрагмент *HTML*-файла. При этом для разных типов блоков могут существовать разные шаблоны.

Для основного блока, описывающего класс и его характеристики, шаблон должен содержать все неинформативные элементы будущей документации. Такими элементами являются: стилевое оформление документации, определение позиции фрагментов информации в документации и т.д.

Например, часть шаблона для основного блока должна содержать заглушки или теги **(**в зависимости от используемого типа шаблона (разд. 4.3 и разд. 5.3)**)**, описывающие названия класса и пакета. В случае использования заглушек шаблон может выглядеть следующим образом:

```
<H2>
   <FONT SIZE="-1">
      #@java@package;
   </FONT>
   <BR>
   #@java@class;
</H2>
```

Шаблон на основе тегов:

```
<H2>
   <FONT SIZE="-1">
      <inetDoc:write property="package" />
   </FONT>
   <BR>
   <inetDoc:write property="classname" />
</H2>
```

После заполнения шаблонов для каждого блока выполняется их объединение в файлы документации.

Шаблоны документации к *Java*-файлам приведены в приложениях 4 и 5.

7.4. Результат генерации

Результатом автоматической генерации является документация (рис. 46).

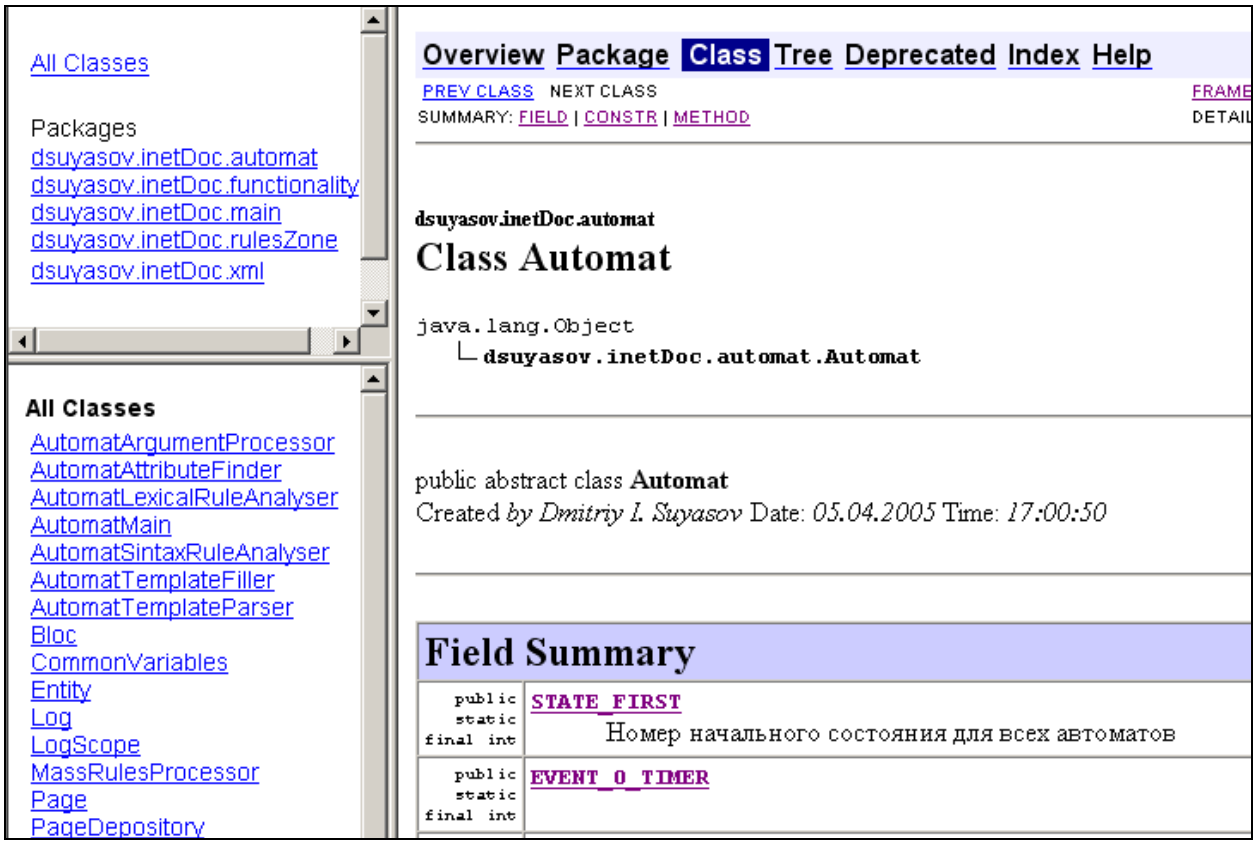

Рис. 46. Внешний вид автоматически сгенерированной документации к *Java*-проекту

Как следует из рисунка, документация содержит описания каждого элемента конструкции языка и множество гипертекстовых ссылок, определяющих связи элементов документируемого класса.

ГЛАВА 8. СРАВНЕНИЕ МЕТОДОВ И ИХ ПЕРСПЕКТИВЫ

8.1. Результаты работы

На основе разработанных методов созданы три программы по автоматическому документированию программных проектов, которые описаны в предыдущей главе. Исходные коды этих программ приведены на сайте http://is.ifmo.ru, раздел «Проекты». Особенностями этих программ являются быстрота, высокое качество и ориентированность на пользователя при создании документации.

Программы базируются на автоматном подходе. Они являются изоморфным отображением спроектированных автоматов. Структура автоматов соответствует структурной схеме, приведенной на рис. 3.

С помощью автоматов решается несколько задач. Часть автоматов управляет работой программы и запуском остальных автоматов. Другая часть автоматов производит разбор документируемых файлов, определяет в них специальные блоки комментариев, анализирует текст и выделяет из него необходимые фрагменты информации. Автоматы заполнения шаблонов получают на вход выделенные фрагменты информации и файлы шаблонов, заполняют их и формируют на выходе файлы документации.

Для работы программ по документированию программных проектов необходимы файлы настроек. Эти файлы создаются пользователями программ. Они содержат правила поиска фрагментов информации в документируемых файлах и шаблоны файлов документации.

В работе в приведены два класса примеров, первый из которых предназначен для документирования Интернет-сайтов, а второй – для документирования проектов, реализованных на языке программирования *Java*. В каждом классе рассмотрены примеры применения всех трех методов, описанных в настоящей работе.

Примеры второго класса свидетельствуют о том, что создаваемая на основе предлагаемых подходов документация по стилю и содержанию может быть аналогична документации, которая создается инструментом *JavaDoc* компании *Sun*.

Разработанные методы автоматического документирования пригодны для создания документации к проектам, включающим в себя разные форматы файлов. В этом смысле эти методы обобщают достижения нескольких программных продуктов, к которым относятся *JavaDoc*, *Doclets*, *xDoclet*, *DTDDoc* и т.д.

Данный проект реализован в рамках «Движения за открытую проектную документацию» [1]. Создание документации к проекту являлось его неотъемлемой частью.

8.2. Сравнение методов

В процессе проектирования систем автоматического документирования программных проектов было разработано три метода. Все они решают поставленную задачу, но каждый из них обладает определенными возможностями и ограничениями. Ниже перечислены основные отличия методов.

1. Все три метода реализованы на основе автоматного подхода. Их отличия заключаются в перечне задач, решаемых автоматами. В методе, основанном на «гибких» правилах, количество автоматов меньше, а функциональность шире, нежели в остальных методах.

В методах на основе регулярных выражений, наоборот, автоматы решают узкие задачи, а их количество больше. При этом отметим, что проектирование автоматов и их связей проще при их малом количестве, а узость выполняемых ими задач позволяет их повторно использовать.

Например, во втором методе (разд. 5) реализованы автоматы синтаксического и лексического разбора выражений. Эти автоматы используются не только при разборе правил поиска информации в документируемых файлах, но и при заполнении шаблонов, в некоторых командах которых применяются эти выражения.

2. Структура правил, на основе которых ведется поиск информации в документируемых файлах, различается в рассматриваемых методах. Первый метод использует «гибкие» правила, а остальные методы используют правила, построенные на основе регулярных выражений. Если «гибкие» правила интуитивно понятны для решения поставленной в работе задачи, то теоретически разработанными и общеизвестными являются регулярные выражения.

3. Принцип построения и структура шаблонов отличается в разных методах. В первом методе в качестве команд, находящихся внутри шаблонов, применяются специально разработанные конструкции. Они отвечают требованиям задания и реализуют функциональность по созданию внешнего вида документации проектов. В остальных методах используются теги *XML* определенного пространства имен. Поэтому в первом случае команды имеют простую структуру, но меньшую функциональность.

4. Принципы разбора регулярных выражений во втором (глава 5) и третьем (глава 6) методах отличаются между собой. Во втором методе проверка текста документируемого файла на основе регулярного выражения производится полностью с использованием автоматного подхода, описанного в книгах [3, 8]. В третьем методе регулярные выражения переводятся в классические автоматы, как описано в главе 6.

8.3. Перспективы

Разработанные методы позволили создать программы для автоматического документирования программных проектов. В качестве примеров для сравнения эффективности методов выполнено документирование проектов Интернет-сайтов и *Java* проектов.

Пользователь программ может дополнительно добавить или модифицировать файлы настроек. Этим он может изменить внешний вид и структуру создаваемой документации. Аналогичным образом добавляется функциональность по документированию проектов на других языках программирования. Таким образом, решаемая задача шире, чем у схожих продуктов [2, 4].

Реализуемые в работе принципы выделения информации из файлов на основе «гибких» правил или регулярных выражений могут быть полезны в задачах анализа содержимого разных форматов файлов или проектов. В дальнейшем выделенная информация может быть использована для составления графиков, отчетов, новостной рассылки и т.д.

Задачи автоматической генерации документации могут быть легко преобразованы в генерацию файлов проекта. Схожую функциональность имеют некоторые продукты, такие как, например, инструменты *xDoclet* [16] и *XSTL* [6]. *XDoclet* создает часть файлов языка программирования *Java* вместо разработчика на этапе создания проекта. *XSLT* на основе файла стилей преобразует *XML*-документы в другие файлы. Реализованные в данной работе методы могут конкурировать с *xDoclet* и *XSLT*, так как является более универсальным и общим средством для генерации файлов для разных языков программирования.

ЗАКЛЮЧЕНИЕ

Создание качественной проектной документации – важная задача любого проекта независимо от его сложности. Процесс создания такой документации является творческим и сложным. Облегчить его может система автоматического документирования, которая позволяет взять часть работы на себя.

В настоящей работе реализованы три метода автоматического документирования программных проектов. Их основу составляют файлы проектов со специальными комментариями, которые создаются разработчиком во время реализации. С помощью комментариев, правил выделения информации внутри файлов, а также шаблонов предложенные методы реализуют задачу автоматического документирования. В результате может быть реализовано описанное Д. Кнутом в книге «Грамотное программирование» [17] совмещение исходного текста программ и документации.

Отличия методов друг от друга заключаются в их внутренней структуре и в формате загружаемых правил поиска информации и шаблонов. При этом каждый метод выполняет поставленную в работе задачу.

Общим для всех методов является автоматный подход. Он позволил создать четкую структуру для каждого из них, формализовать методы независимо от языка программирования, упростить задачу документирования проекта. Автоматный подход способствовал легкости реализации и быстрой отладке всей системы.

На основе разработанных методов созданы три программы автоматического документирования, а также файлы настроек для документирования Интернет-сайтов и проектов, реализованных на языке программирования *Java*. Файлы настроек для других языков

программирования могут быть реализованы пользователем дополнительно без изменения программ.

Источники

1. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию //PC Week/RE. 2003. № 40, с. 38, 39, 42. http://is.ifmo.ru/works/open_doc/
2. *Javadoc Tool*. http://java.sun.com/j2se/javadoc
3. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
4. *DTD Documentation Tool*. http://dtddoc.sourceforge.net
5. *Питц-Моултис Н., Кирк Ч.* XML в подлиннике. СПб.: БХВ-Петербург, 2001.
6. *Холзнер С.* XSLT библиотека программиста. СПб.: издательский дом «Питер», 2002.
7. *Ломов А. Ю.* HTML, CSS, скрипты: практика создания сайтов. СПб: БХВ-Петербург, 2005.
8. *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации. СПб.: Наука, 2000. http://is.ifmo.ru/books/log_upr/1
9. *Шалыто А.А., Туккель Н.И.* Проектирование программного обеспечения системы управления дизель-генераторами на основе автоматного подхода. СПб.: ФГУП "НПО "Аврора". 2002. Вып. 5, с. 66-82. http://is.ifmo.ru/works/diesel/
10. *Дорот В.Л., Новиков Ф.А.* Толковый словарь современной компьютерной лексики. СПб.: БХВ-Петербург, 1999.
11. *Нефедов В.Н., Осипова В.А.* Курс дискретной математики. М.: МАИ, 1992.

12. *Фридл Д.* Регулярные выражения. СПб.: Питер, 2003.
13. *World Wide Web Консорциум.* Пространства имен в XML. 14.01.1999. http://www.w3.org/TR/REC-xml-names
14. *Ахо А., Сети Р., Ульман Дж.* Компиляторы. Принципы, технологии, инструменты. М.: Вильямс, 2003.
15. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 1990.
16. *xDoclet Java tool*. http://sourceforge.net/projects/xdoclet
17. *Knuth D.E.* Literate Programming. CSLI Lecture Notes. 1992. № 27. Stanford: Center for the Study of Language and Information.

Приложение 1. Пример исходного текста класса языка программирования *Java*

```
/*@java
 @Created by Dmitriy I. Suyasov
 @Date 05.04.2005
 @Time 17:00:50
 */
package dsuyasov.inetDoc.automat;

import dsuyasov.inetDoc.functionality.Log;

public abstract class Automat {

    /*@
        Номер начального состояния для всех автоматов
    */
    public static final int STATE_FIRST = 0;
    public static final int EVENT_0_TIMER = 0;

    protected String automatName;

    /*@
        Конструктор.
        @param automatNumber Номер автомата
    */
    public Automat(String automatName) {
        y_state = STATE_FIRST;
        this.automatName = automatName;
    }

    /*@
        Функция запуска автомата
    */
    protected void work() {
        log.automat(automatName, true);
        running = true;
        ERROR = "";
        y_state = STATE_FIRST;
        initialize();
        while (running) {
            automat(EVENT_0_TIMER);
        }
        if (ERROR.length() > 0)
            ERROR = "["+this.getClass().getName()+"] ->\n"+ERROR;
        log.automat(automatName, false);
    }

    /*@
        Функция работы автомата
        @param event Номер пришедшего события
    */
    protected abstract void automat(int event);

    /*@
        Функция инициализации автомата
    */
    protected abstract void initialize();
```

```
    /*@
        Переменная-индикатор работы автомата
    */
    public boolean running = true;
    /*@
        Номер состояния, в котором находится автомат
    */
    protected int y_state;

    /*@
        Объект для протоколирования
    */
    protected Log log = Log.getLogger();

    public String ERROR = "";

}
```

Приложение 2. Пример описания правил поиска информации в *Java*-файле на основе «гибких»

```
<?xml version="1.0" encoding="Windows-1251"?>
<template>
      <defaultentity name="bloc" />
  <baseentity>
    <defaultattribute name="defaultbloc"/>
    <attribute name="defaultbloc"/>
    <template-file href="templates\template_java_base.html"/>
  </baseentity>

  <headentity name="java">
    <defaultattribute name="Description"/>
    <attribute name="Created"/>
    <attribute name="User"/>
    <attribute name="Date"/>
    <attribute name="Description"/>
    <attribute name="Time"/>
    <attribute name="class"/>
    <attribute name="function"/>
    <attribute name="field"/>
    <attribute name="package"/>
    <attribute name="varInit"/>
    <attribute name="varInitName"/>
    <attribute name="varNotInit"/>
    <template-file href="templates\template_java_java.html"/>
    <rules>

      <rule name="package">
        <start><![CDATA[!package|!space]]></start>
        <end><![CDATA[!sep]]></end>
        <def name="package"><![CDATA[package]]></def>
      </rule>

      <rule name="class">
        <start>
<![CDATA[!public|P!space|?abstract|*space|!class,interface|*space|!ex(space)]
]>
        </start>
        <end><![CDATA[!space]]></end>
        <def name="class"><![CDATA[class]]></def>
        <def name="abstract"><![CDATA[abstract]]></def>
        <def name="interface"><![CDATA[interface]]></def>
      </rule>

      <rule name="comment1">
        <start><![CDATA[!copen]]></start>
        <end><![CDATA[!cclose]]></end>
        <def name="copen"><![CDATA[/*]]></def>
        <def name="cclose"><![CDATA[*/]]></def>
      </rule>

      <rule name="comment2">
        <start><![CDATA[!copen]]></start>
        <end><![CDATA[!cclose]]></end>
        <def name="copen"><![CDATA[//]]></def>
        <def name="cclose"><![CDATA[\n]]></def>
```

```
</rule>

<rule name="inclass">
  <start><![CDATA[!bopen]]></start>
  <end><![CDATA[!bclose]]></end>

  <rule name="comment1">
    <start><![CDATA[!copen]]></start>
    <end><![CDATA[!cclose]]></end>
    <def name="copen"><![CDATA[/*]]></def>
    <def name="cclose"><![CDATA[*/]]></def>
  </rule>

  <rule name="comment2">
    <start><![CDATA[!copen]]></start>
    <end><![CDATA[!cclose]]></end>
    <def name="copen"><![CDATA[//]]></def>
    <def name="cclose"><![CDATA[\n]]></def>
  </rule>

  <rule name="field">
    <start><![CDATA[P!public,protected,private|!space]]></start>
    <end><![CDATA[P!bracket,sep|!ex(bracket)|*ex(bracket)]]></end>
    <def name="bracket"><![CDATA[)]]></def>
    <def name="bracketopen"><![CDATA[(]]></def>

    <rule name="function">
      <start>
      <![CDATA[P!public,protected,private|*ex(equal)|!bracketopen]]>
      </start>
      <end><![CDATA[P!bracket]]></end>
    </rule>

    <rule name="varInit">
      <start>
        <![CDATA[P!public,protected,private|*ex(equal)|!equal]]>
      </start>
      <end><![CDATA[P!sep]]></end>

      <rule name="varInitName">
        <start>
<![CDATA[P!public,protected,private|*ex(equal)|!equal|!ex(equal)]]>
        </start>
        <end><![CDATA[!equal|*ex(equal)]]></end>
      </rule>

      <replace>
        <from><![CDATA["]]></from>
        <to><![CDATA[\\"]]></to>
      </replace>

    </rule>

    <rule name="varNotInit">
      <start>
        <![CDATA[P!public,protected,private|*ex(equal)|!sep]]>
      </start>
      <end><![CDATA[P!sep|?ex(sep)]]></end>
    </rule>
  </rule>
```

```
        <rule name="inc1">
          <start><![CDATA[!bopen]]></start>
          <end><![CDATA[!bclose]]></end>

          <rule name="inc2">
            <start><![CDATA[!bopen]]></start>
            <end><![CDATA[!bclose]]></end>

            <rule name="inc3">
              <start><![CDATA[!bopen]]></start>
              <end><![CDATA[!bclose]]></end>

              <rule name="inc4">
                <start><![CDATA[!bopen]]></start>
                <end><![CDATA[!bclose]]></end>

                <rule name="inc5">
                  <start><![CDATA[!bopen]]></start>
                  <end><![CDATA[!bclose]]></end>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>

      </rule>

      <def name="quote"><![CDATA["]]></def>
      <def name="quote2"><![CDATA[\\]]></def>
      <def name="equal"><![CDATA[=]]></def>
      <def name="space"><![CDATA[ ]]></def>
      <def name="name"><![CDATA[name]]></def>
      <def name="public"><![CDATA[public]]></def>
      <def name="protected"><![CDATA[protected]]></def>
      <def name="private"><![CDATA[private]]></def>
      <def name="bopen"><![CDATA[{]]></def>
      <def name="bclose"><![CDATA[}]]></def>
      <def name="sep"><![CDATA[;]]></def>
    </rules>
  </headentity>

  <entity name="bloc">
    <defaultattribute name="description"/>
    <attribute name="description"/>
    <attribute name="function"/>
    <attribute name="field"/>
    <attribute name="package"/>
    <attribute name="varInit"/>
    <attribute name="varInitName"/>
    <attribute name="varNotInit"/>
    <attribute name="param"/>
    <attribute name="return"/>
    <template-file href="templates\template_java_bloc.html"/>
    <rules>
      <def name="public"><![CDATA[public]]></def>
      <def name="protected"><![CDATA[protected]]></def>
      <def name="private"><![CDATA[private]]></def>
      <def name="sep"><![CDATA[;]]></def>
      <def name="bopen"><![CDATA[{]]></def>
      <def name="bclose"><![CDATA[}]]></def>
      <def name="space"><![CDATA[ ]]></def>
```

```
<def name="equal"><![CDATA[=]]></def>

<rule name="comment1">
  <start><![CDATA[!copen]]></start>
  <end><![CDATA[!cclose]]></end>
  <def name="copen"><![CDATA[/*]]></def>
  <def name="cclose"><![CDATA[*/]]></def>
</rule>

<rule name="comment2">
  <start><![CDATA[!copen]]></start>
  <end><![CDATA[!cclose]]></end>
  <def name="copen"><![CDATA[//]]></def>
  <def name="cclose"><![CDATA[\n]]></def>
</rule>

<rule name="field">
  <start><![CDATA[P!public,protected,private|!space]]></start>
  <end><![CDATA[P!bracket,sep|!ex(bracket)|*ex(bracket)]]></end>

  <def name="bracket"><![CDATA[)]]></def>
  <def name="bracketopen"><![CDATA[(]]></def>

  <rule name="function">
    <start>
      <![CDATA[P!public,protected,private|*ex(equal)|!bracketopen]]>
    </start>
    <end><![CDATA[P!bracket]]></end>
  </rule>

  <rule name="varInit">
    <start>
      <![CDATA[P!public,protected,private|*ex(equal)|!equal]]>
    </start>
    <end><![CDATA[P!sep]]></end>

    <rule name="varInitName">
      <start>
  <![CDATA[P!public,protected,private|*ex(equal)|!equal|!ex(equal)]]>
      </start>
      <end><![CDATA[!equal|*ex(equal)]]></end>
    </rule>

    <replace>
      <from><![CDATA["]]></from>
      <to><![CDATA[\\"]]></to>
    </replace>

  </rule>

  <rule name="varNotInit">
    <start>
      <![CDATA[P!public,protected,private|*ex(equal)|!sep]]>
    </start>
    <end><![CDATA[P!sep|?ex(sep)]]></end>
  </rule>

  <replace>
    <from><![CDATA["]]></from>
    <to><![CDATA[\\"]]></to>
  </replace>
```

```
      </rule>

      <rule name="inc1">

        <start><![CDATA[!bopen]]></start>
        <end><![CDATA[!bclose]]></end>

        <rule name="inc2">
          <start><![CDATA[!bopen]]></start>
          <end><![CDATA[!bclose]]></end>

          <rule name="inc3">
            <start><![CDATA[!bopen]]></start>
            <end><![CDATA[!bclose]]></end>

            <rule name="inc4">
              <start><![CDATA[!bopen]]></start>
              <end><![CDATA[!bclose]]></end>

              <rule name="inc5">
                <start><![CDATA[!bopen]]></start>
                <end><![CDATA[!bclose]]></end>
              </rule>
            </rule>
          </rule>
        </rule>
      </rule>

      <rule name="outOfClass">
        <start><![CDATA[!bclose]]></start>
        <end><![CDATA[!bopen]]></end>

        <rule name="comment1">
          <start><![CDATA[!copen]]></start>
          <end><![CDATA[!cclose]]></end>
          <def name="copen"><![CDATA[/*]]></def>
          <def name="cclose"><![CDATA[*/]]></def>
        </rule>

        <rule name="comment2">
          <start><![CDATA[!copen]]></start>
          <end><![CDATA[!cclose]]></end>
          <def name="copen"><![CDATA[//]]></def>
          <def name="cclose"><![CDATA[\n]]></def>
        </rule>

      </rule>
    </rules>
  </entity>

  <rules>

    <def name="comment"><![CDATA[/*]]></def>
    <def name="command"><![CDATA[@]]></def>
    <def name="end"><![CDATA[*/]]></def>
    <def name="space"><![CDATA[ ]]></def>
    <def name="newline"><![CDATA[\r\n]]></def>

    <rule name="java">
      <start><![CDATA[!comment|!command|!javatext]]></start>
```

```
   <end><![CDATA[!end]]></end>
   <def name="javatext"><![CDATA[java]]></def>
</rule>

<rule name="comment">
   <start><![CDATA[!comment|P!ex(command)]]></start>
   <end><![CDATA[!end]]></end>
</rule>

<rule name="comment2">
   <start><![CDATA[!copen]]></start>
   <end><![CDATA[!cclose]]></end>
   <def name="copen"><![CDATA[//]]></def>
   <def name="cclose"><![CDATA[\n]]></def>
</rule>


<rule name="bracket">
   <start><![CDATA[!br1]]></start>
   <end><![CDATA[!br2]]></end>

   <def name="br1"><![CDATA[{]]></def>
   <def name="br2"><![CDATA[}]]></def>

   <rule name="comment">
      <start><![CDATA[!comment|P!ex(command)]]></start>
      <end><![CDATA[!end]]></end>
   </rule>

   <rule name="comment2">
      <start><![CDATA[!copen]]></start>
      <end><![CDATA[!cclose]]></end>
      <def name="copen"><![CDATA[//]]></def>
      <def name="cclose"><![CDATA[\n]]></def>
   </rule>

   <rule name="bloc">
      <start><![CDATA[!comment|!command|!bloctext,space,newline]]></start>
      <end><![CDATA[!end]]></end>
      <def name="bloctext"><![CDATA[bloc ]]></def>
   </rule>

   <rule name="inc1">
      <start><![CDATA[!br1]]></start>
      <end><![CDATA[!br2]]></end>

      <rule name="inc2">
         <start><![CDATA[!br1]]></start>
         <end><![CDATA[!br2]]></end>

         <rule name="inc3">
            <start><![CDATA[!br1]]></start>
            <end><![CDATA[!br2]]></end>

            <rule name="inc4">
               <start><![CDATA[!br1]]></start>
               <end><![CDATA[!br2]]></end>

               <rule name="inc5">
                  <start><![CDATA[!br1]]></start>
                  <end><![CDATA[!br2]]></end>
```

```
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>
      </rule>

    </rules>
  </template>
```

Приложение 3. Пример описания правил поиска информации в *Java*-файле на основе регулярных выражений

```
<?xml version="1.0" encoding="Windows-1251"?>
<template>
  <defaultentity name="java" />
  <template-file href="templates\template_java_base.html"/>

  <entity name="java">
    <defaultattribute name="Description"/>
    <attribute name="Created"/>
    <attribute name="User"/>
    <attribute name="Date"/>
    <attribute name="Description"/>
    <attribute name="Time"/>
    <attribute name="classname"/>
    <attribute name="classtype"/>
    <attribute name="function"/>
    <attribute name="functionname"/>
    <attribute name="functioninside"/>
    <attribute name="import"/>
    <attribute name="extends"/>
    <attribute name="implements"/>
    <attribute name="package"/>
    <attribute name="element"/>
    <attribute name="var"/>
    <attribute name="varname"/>
    <attribute name="varequal"/>
    <template-file href="templates\template_java_java.html"/>
    <rules>

      <rule name="package">
        <start><![CDATA[package[\s\n\t\r]+\P\w]]></start>
        <end><![CDATA[;]]></end>
      </rule>

      <rule name="import">
        <start><![CDATA[import[\s\n\t\r]+\P\w]]></start>
        <end><![CDATA[;]]></end>
      </rule>

      <rule name="class">
        <start>
    <![CDATA[(public)?[\s\r\t\n]*(abstract)?[\s\r\t\n]*(class|interface)]]>
        </start>
        <end><![CDATA[\}]]></end>

        <rule name="classtype">
          <start>
    <![CDATA[(public)?[\s\r\t\n]*(abstract)?[\s\r\t\n]*(class|interface)]]>
          </start>
          <end><![CDATA[[\n\t\r\s]]]></end>
        </rule>
```

```
            <rule name="classname">
              <start><![CDATA[(class|interface)[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="extends">
              <start><![CDATA[extends[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="implements">
              <start><![CDATA[implements[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="inclass">
              <start><![CDATA[\{.\P]]></start>
              <end><![CDATA[\}]]></end>

              <rule name="comment1">
                <start><![CDATA[/\*\P]]></start>
                <end><![CDATA[\*/\P]]></end>
              </rule>
              <rule name="comment2">
                <start><![CDATA[//]]></start>
                <end><![CDATA[\n\P]]></end>
              </rule>

              <rule name="element">
                <start><![CDATA[\w]]></start>
                <end><![CDATA[[;\)]]]></end>
                <rule name="function" priority="0">
                  <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\n]+\w+[\s\n\t\n]*\(]]>
                  </start>
                  <end><![CDATA[\)\P]]></end>

                  <rule name="functionname" priority="0">
                    <start><![CDATA[[\s\n\t\n]+\P\w+[\s\n\t\n]*\(]]></start>
                    <end><![CDATA[[\s\n\t\n]*\(]]></end>
                  </rule>

                  <rule name="functioninside" priority="0">
                    <start><![CDATA[\(\P(\n|.)]]></start>
                    <end><![CDATA[\)]]></end>
                  </rule>
                </rule>

                <rule name="var" priority="1">
                  <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\n]+\w+[\s\n\t\n]*[=;]]]>
                  </start>
                  <end><![CDATA[;\P]]></end>

                  <rule name="varname">
                    <start>
                      <![CDATA[[^\s\t\r\n=][\s\r\t\n]+\P\w+[\s\n\t\n]*[=;]]]>
                    </start>
                    <end><![CDATA[[;=]]]></end>
```

```
            </rule>
          </rule>

          <rule name="inc1">
            <start><![CDATA[\{\P]]></start>
            <end><![CDATA[\}.\P]]></end>

            <rule name="inc2">
              <start><![CDATA[\{\P]]></start>
              <end><![CDATA[\}.\P]]></end>

              <rule name="inc3">
                <start><![CDATA[\{\P]]></start>
                <end><![CDATA[\}.\P]]></end>

                <rule name="inc4">
                  <start><![CDATA[\{\P]]></start>
                  <end><![CDATA[\}.\P]]></end>

                  <rule name="inc5">
                    <start><![CDATA[\{\P]]></start>
                    <end><![CDATA[\}.\P]]></end>
                  </rule>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>

      </rule>

      <rule name="comment1">
        <start><![CDATA[/\*\P]]></start>
        <end><![CDATA[\*/]]></end>
      </rule>

      <rule name="comment2">
        <start><![CDATA[//]]></start>
        <end><![CDATA[\n]]></end>
      </rule>

    </rules>
  </entity>

  <entity name="bloc">
    <defaultattribute name="description"/>
    <attribute name="description"/>
    <attribute name="function"/>
    <attribute name="functionname"/>
    <attribute name="functioninside"/>
    <attribute name="param"/>
    <attribute name="return"/>
    <attribute name="var"/>
    <attribute name="varname"/>
    <attribute name="deprecated"/>
    <attribute name="element"/>
    <template-file href="templates\template_java_bloc.html"/>
    <rules>

      <rule name="comment1">
```

```
          <start><![CDATA[/\*\P]]></start>
          <end><![CDATA[\*/\P]]></end>
        </rule>


        <rule name="comment2">
          <start><![CDATA[//]]></start>
          <end><![CDATA[\n\P]]></end>
        </rule>

        <rule name="element" endpriority="0">
          <start><![CDATA[\w]]></start>
          <end><![CDATA[[;\)](\n|.)]]></end>

          <rule name="function" priority="0">
            <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\r]+\w+[\s\n\t\r]*\(]]>
            </start>
            <end><![CDATA[\)\P]]></end>

            <rule name="functionname" priority="0">
              <start><![CDATA[[\s\n\t\n]+\P\w+[\s\n\t\n]*\(]]></start>
              <end><![CDATA[[\s\n\t\n]*\(]]></end>
            </rule>

            <rule name="functioninside" priority="0">
              <start><![CDATA[\(\P(\n|.)]]></start>
              <end><![CDATA[\)]]></end>
            </rule>
          </rule>

          <rule name="var" priority="1">
            <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\r]+\w+[\s\n\t\r]*[=;]]]>
            </start>
            <end><![CDATA[;\P]]></end>

            <rule name="varname">
              <start>
                <![CDATA[[^\s\t\r\n=][\s\r\t\n]+\P\w+[\s\n\t\n]*[=;]]]>
              </start>
              <end><![CDATA[[;=]]]></end>
            </rule>
          </rule>

          <rule name="inc1">
            <start><![CDATA[\{\P]]></start>
            <end><![CDATA[\}.\P]]></end>

            <rule name="inc2">
              <start><![CDATA[\{\P]]></start>
              <end><![CDATA[\}.\P]]></end>

              <rule name="inc3">
                <start><![CDATA[\{\P]]></start>
                <end><![CDATA[\}.\P]]></end>

                <rule name="inc4">
                  <start><![CDATA[\{\P]]></start>
```

```
                <end><![CDATA[\}.\P]]></end>

                <rule name="inc5">
                  <start><![CDATA[\{\P]]></start>
                  <end><![CDATA[\}.\P]]></end>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>

        <rule name="inc1">
          <start><![CDATA[\{\P]]></start>
          <end><![CDATA[\}.\P]]></end>

          <rule name="inc2">
            <start><![CDATA[\{\P]]></start>
            <end><![CDATA[\}.\P]]></end>

            <rule name="inc3">
              <start><![CDATA[\{\P]]></start>
              <end><![CDATA[\}.\P]]></end>

              <rule name="inc4">
                <start><![CDATA[\{\P]]></start>
                <end><![CDATA[\}.\P]]></end>

                <rule name="inc5">
                  <start><![CDATA[\{\P]]></start>
                  <end><![CDATA[\}.\P]]></end>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>
      </rules>
    </entity>

    <entity name="class">
      <defaultattribute name="description"/>
      <attribute name="description"/>
      <attribute name="deprecated"/>
      <attribute name="classname"/>
      <attribute name="classtype"/>
      <attribute name="function"/>
      <attribute name="functionname"/>
      <attribute name="functioninside"/>
      <attribute name="extends"/>
      <attribute name="implements"/>
      <attribute name="element"/>
      <attribute name="var"/>
      <attribute name="varname"/>
      <attribute name="varequal"/>
      <template-file href="templates\template_java_java.html"/>

      <rules>

        <rule name="class">
          <start>
      <![CDATA[(public)?[\s\r\t\n]*(abstract)?[\s\r\t\n]*(class|interface)]]>
```

```
            </start>
            <end><![CDATA[\}]]></end>

            <rule name="classtype">
              <start>
  <![CDATA[(public)?[\s\r\t\n]*(abstract)?[\s\r\t\n]*(class|interface)]]>
              </start>
              <end><![CDATA[[\n\t\r\s]]]></end>
            </rule>

            <rule name="classname">
              <start><![CDATA[(class|interface)[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="extends">
              <start><![CDATA[extends[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="implements">
              <start><![CDATA[implements[\n\t\r\s]+\P\w]]></start>
              <end><![CDATA[[\s\t\r\n{]]]></end>
            </rule>

            <rule name="inclass">
              <start><![CDATA[\{.\P]]></start>
              <end><![CDATA[\}]]></end>

              <rule name="comment1">
                <start><![CDATA[/\*\P]]></start>
                <end><![CDATA[\*/\P]]></end>
              </rule>

              <rule name="comment2">
                <start><![CDATA[//]]></start>
                <end><![CDATA[\n\P]]></end>
              </rule>

              <rule name="element">
                <start><![CDATA[\w]]></start>
                <end><![CDATA[[;\)]]]></end>

                <rule name="function" priority="0">
                  <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\n]+\w+[\s\n\t\n]*\(]]>
                  </start>
                  <end><![CDATA[\)\P]]></end>

                  <rule name="functionname" priority="0">
                    <start><![CDATA[[\s\n\t\n]+\P\w+[\s\n\t\n]*\(]]></start>
                    <end><![CDATA[[\s\n\t\n]*\(]]></end>
                  </rule>

                  <rule name="functioninside" priority="0">
                    <start><![CDATA[\(\P(\n|.)]]></start>
                    <end><![CDATA[\)]]></end>
                  </rule>
                </rule>
```

```
            <rule name="var" priority="1">
              <start>
<![CDATA[(public|protected|private)?[\s\n\r\t]*(static|abstract)?[\s\n\r\t]*(
final)?[\s\n\r\t]*(\w|\[|\])+[\s\n\t\n]+\w+[\s\n\t\n]*[=;]]]>
              </start>
              <end><![CDATA[;\P]]></end>

              <rule name="varname">
                <start>
                  <![CDATA[[^\s\t\r\n=][\s\r\t\n]+\P\w+[\s\n\t\n]*[=;]]]>
                </start>
                <end><![CDATA[[;=]]]></end>
              </rule>
            </rule>

            <rule name="inc1">
              <start><![CDATA[\{\P]]></start>
              <end><![CDATA[\}.\P]]></end>

              <rule name="inc2">
                <start><![CDATA[\{\P]]></start>
                <end><![CDATA[\}.\P]]></end>

                <rule name="inc3">
                  <start><![CDATA[\{\P]]></start>
                  <end><![CDATA[\}.\P]]></end>

                  <rule name="inc4">
                    <start><![CDATA[\{\P]]></start>
                    <end><![CDATA[\}.\P]]></end>

                    <rule name="inc5">
                      <start><![CDATA[\{\P]]></start>
                      <end><![CDATA[\}.\P]]></end>
                    </rule>
                  </rule>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>

        <rule name="comment1">
          <start><![CDATA[/\*\P]]></start>
          <end><![CDATA[\*/]]></end>
        </rule>

        <rule name="comment2">
          <start><![CDATA[//]]></start>
          <end><![CDATA[\n]]></end>
        </rule>
      </rules>
    </entity>

    <rules>

      <rule name="java">
        <start><![CDATA[/\*@java[\s\n\t\r]]]></start>
        <end><![CDATA[\*/]]></end>
```

```
    <rule name="Description">
      <start><![CDATA[@([dD]escription)?[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>

    <rule name="Created">
      <start><![CDATA[@[cC]reated[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>

    <rule name="User">
      <start><![CDATA[@User[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>

    <rule name="Date">
      <start><![CDATA[@[dD]ate[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>

    <rule name="Time">
      <start><![CDATA[@[tT]ime[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>
  </rule>

  <rule name="class">
    <start><![CDATA[package(.|\n)+\P/\*@(class)?[\s\t\n\r]]]></start>
    <end><![CDATA[\*/]]></end>

    <rule name="description">
      <start><![CDATA[@([dD]escription)?[\s\r\t\n]+\P\w]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>

    <rule name="deprecated">
      <start><![CDATA[@deprecated[\s\r\t\n]+\P[^\n\r\t\s]]]></start>
      <end><![CDATA[@|\*/]]></end>
    </rule>
  </rule>

  <rule name="comment1">
    <start><![CDATA[/\*[^@]]]></start>
    <end><![CDATA[\*/]]></end>
  </rule>

  <rule name="comment2">
    <start><![CDATA[//]]></start>
    <end><![CDATA[\n]]></end>
  </rule>

  <rule name="bracket">
    <start><![CDATA[\{\P]]></start>
    <end><![CDATA[\}.]]></end>

    <rule name="comment">
      <start><![CDATA[/\*[^@]\P]]></start>
      <end><![CDATA[\*/]]></end>
    </rule>

    <rule name="comment2">
```

```
          <start><![CDATA[//]]></start>
          <end><![CDATA[\n]]></end>
        </rule>

        <rule name="bloc">
          <start><![CDATA[/\*@(bloc)?[\s\n\t\r]]]></start>
          <end><![CDATA[\*/]]></end>

          <rule name="description">
            <start><![CDATA[@([dD]escription)?[\s\r\t\n]+\P\w]]></start>
            <end><![CDATA[@|\*/]]></end>
          </rule>

          <rule name="param">
            <start><![CDATA[@[pP]aram[\s\r\t\n]+\P\w]]></start>
            <end><![CDATA[@|\*/]]></end>
          </rule>

          <rule name="return">
            <start><![CDATA[@[rR]eturn[\s\r\t\n]+\P\w]]></start>
            <end><![CDATA[@|\*/]]></end>
          </rule>

          <rule name="deprecated">
            <start><![CDATA[@deprecated[\s\r\t\n]+\P[^\n\r\t\s]]]></start>
            <end><![CDATA[@|\*/]]></end>
          </rule>
        </rule>

        <rule name="inc1">
          <start><![CDATA[\{\P]]></start>
          <end><![CDATA[\}\P.]]></end>

          <rule name="inc2">
            <start><![CDATA[\{\P]]></start>
            <end><![CDATA[\}\P.]]></end>

            <rule name="inc3">
              <start><![CDATA[\{\P]]></start>
              <end><![CDATA[\}\P.]]></end>

              <rule name="inc4">
                <start><![CDATA[\{\P]]></start>
                <end><![CDATA[\}\P.]]></end>

                <rule name="inc5">
                  <start><![CDATA[\{\P]]></start>
                  <end><![CDATA[\}\P.]]></end>
                </rule>
              </rule>
            </rule>
          </rule>
        </rule>
      </rule>
    </rules>
  </template>
```

Приложение 4. Пример шаблона документации к основному блоку файла *Java*-проекта на основе заглушек

```
<!-- ======== START OF CLASS DATA ======== -->
<H2>
<FONT SIZE="-1">
#@java@package;</FONT>
<BR>
#@java@class;</H2>
<HR>
<DL>
<DT><script language="javascript">
document.writeln(getNotClassName());
document.writeln('<b>'+getClassName()+'</b>');
</script></DL>

<P>
#@if#java@Created;Created #@java@Created;#@ifEnd#java@Created;
#@if#java@User; User: #@java@User;#@ifEnd#java@User;
#@if#java@Date; Date: #@java@Date;#@ifEnd#java@Date;
#@if#java@Time; Time: #@java@Time;#@ifEnd#java@Time;
<P>

<P><HR>
<script language="javascript">
  function isConstructor(str) {
     str = str.substring(0, str.indexOf("("));
     while ((str.lastIndexOf(" ")==str.length-1) ||
(str.lastIndexOf("\r")==str.length-1) || (str.lastIndexOf("\n")==str.length-
1) || (str.lastIndexOf("\t")==str.length-1))
        str = str.substring(0, str.length-1);
     pos = str.lastIndexOf(getClassName());
     symbolStr = "\n\r\t (";
     if ((pos>=0) && (symbolStr.indexOf(str.charAt(pos-1))>=0) &&
(pos+getClassName().length==str.length))//(symbolStr.indexOf(str.charAt(pos+g
etClassName().length))>=0))
        return true;
     return false;
  }
</script><P>

<!-- =========== FIELD SUMMARY =========== -->

<A NAME="field_summary"><!-- --></A><TABLE BORDER="1" WIDTH="100%"
CELLPADDING="3" CELLSPACING="0" SUMMARY="" id="fieldSummary"><tbody
id="field_summary_table">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
<TD COLSPAN=2><FONT SIZE="+2">
<B>Field Summary</B></FONT></TD>
</TR>
#@repeat#java@varNotInit;
<TR BGCOLOR="white" CLASS="TableRowColor" id="varNotInit#@repeatIndex;">
<TD ALIGN="right" VALIGN="top" WIDTH="1%"><FONT SIZE="-1">
<CODE> 
<script language="javascript">
```

```
   str = "#@java@varNotInit;";
   str = str.substring(0, str.length-1);
   while (str.lastIndexOf(' ')==str.length-1) str = str.substring(0,
str.length-1);
   name = str.substring(str.lastIndexOf(' ')+1);
   str = str.substring(0, str.lastIndexOf(' '));
   if (str.indexOf('public')==0) {
      str = str.substring(7);
      while (str.indexOf(' ')==0) str = str.substring(1);
   }
   document.write(str);
</script>
</CODE></FONT></TD>
<TD><CODE><B><A href="#varNotInitDetail#@repeatIndex;">
<script language="javascript">
   document.write(name);
</script>
</A></B></CODE>

<BR>
           </TD>
</TR>
#@repeatEnd#java@varNotInit;

#@repeat#java@varInitName;
<TR BGCOLOR="white" CLASS="TableRowColor" id="varInit#@repeatIndex;">
<TD ALIGN="right" VALIGN="top" WIDTH="1%"><FONT SIZE="-1">
<CODE> 
<script language="javascript">
   str = "#@java@varInitName;";
   str = str.substring(0, str.length-1);
   while (str.lastIndexOf(' ')==str.length-1) str = str.substring(0,
str.length-1);
   name = str.substring(str.lastIndexOf(' ')+1);
   str = str.substring(0, str.lastIndexOf(' '));
   if (str.indexOf('public')==0) {
      str = str.substring(7);
      while (str.indexOf(' ')==0) str = str.substring(1);
   }
   document.write(str);
</script>
</CODE></FONT></TD>
<TD><CODE><B><A HREF="#varInitDetail#@repeatIndex;">
<script language="javascript">
   document.write(name);
</script>
</A></B></CODE>

<BR>
           </TD>
</TR>
#@repeatEnd#java@varInitName;
</tbody>
</TABLE> 

<!-- ======== CONSTRUCTOR SUMMARY ======== -->

<A NAME="constructor_summary"><!-- --></A><TABLE BORDER="1" WIDTH="100%"
CELLPADDING="3" CELLSPACING="0" SUMMARY="" id="constructorSummary"><tbody
id="constructor_summary_table">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
```

```
<TD COLSPAN=2><FONT SIZE="+2">
<B>Constructor Summary</B></FONT></TD></TR>
#@repeat#java@function;
<TR BGCOLOR="white" CLASS="TableRowColor" id="cons#@repeatIndex;">
<TD><CODE><B><A HREF="#consDetail#@repeatIndex;">
<script language="javascript">
  str = "#@java@function;";
  pos = str.indexOf(getClassName());
  symbolStr = "\n\r\t (";
  if (!isConstructor(str))
    cons#@repeatIndex;.style.display = 'none';
  type = str.substring(0, str.indexOf(' '));
  str = str.substring(str.indexOf(' '));
  while (str.indexOf(' ')==0) str = str.substring(1);
  document.write(str);
</script>
</A></B></CODE>
<BR>           </TD>
</TR>
#@repeatEnd#java@function;
</tbody>
</TABLE> 

<!-- ========== METHOD SUMMARY =========== -->

<A NAME="method_summary"><!-- --></A>
<TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
id="methodSummary"><tbody id="method_summary_table">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
<TD COLSPAN=2><FONT SIZE="+2">
<B>Method Summary</B></FONT></TD>
</TR>
#@repeat#java@function;
<TR BGCOLOR="white" CLASS="TableRowColor" id="func#@repeatIndex;">
<TD ALIGN="right" VALIGN="top" WIDTH="1%"><FONT SIZE="-1">
<CODE>
<script language="javascript">
  str = "#@java@function;";
  inside = str.substring(str.indexOf('('));
  tmp = str.substring(0, str.indexOf('('));
  while (tmp.lastIndexOf(' ')==tmp.length-1) tmp = tmp.substring(0,
tmp.length-1);
  index = tmp.lastIndexOf(' ');
  tmp2 = str.substring(0, index);
  name = tmp.substring(index);

  pos = str.indexOf(getClassName());
  symbolStr = "\n\r\t (";
  if (isConstructor(str))
    func#@repeatIndex;.style.display = 'none';
  if (tmp2.indexOf('public')==0) {
    tmp2 = tmp2.substring(tmp2.indexOf('public')+7);
    while (tmp2.indexOf(' ')==0) tmp2 = tmp2.substring(1);
  }
  document.write(tmp2);
</script>
</CODE></FONT></TD>
<TD><CODE><B><A HREF="#funcDetail#@repeatIndex;">
<script language="javascript">
  document.write(name+inside);
</script>
```

```
</A></B></CODE>

<BR>
          </TD>
</TR>
#@repeatEnd#java@function;
</tbody>
</TABLE> 
<P><a name="rule"/>

<!-- ============ FIELD DETAIL =========== -->

<A NAME="field_detail"><!-- --></A>
<TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
id="fieldDetail">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
<TD COLSPAN=1><FONT SIZE="+2">
<B>Field Detail</B></FONT></TD>
</TR>
</TABLE>

<span id="field_detail_text">
#@repeat#java@varNotInit;
<A NAME="#varNotInitDetail#@repeatIndex;"><!-- --></A><H3>
<script language="javascript">
   str = "#@java@varNotInit;";
   str = str.substring(0, str.length-1);
   while (str.lastIndexOf(' ')==str.length-1) str = str.substring(0,
str.length-1);
   name = str.substring(str.lastIndexOf(' ')+1);
   str = str.substring(0, str.lastIndexOf(' '));
   document.write(name);
</script>
</H3>
<PRE>
<script language="javascript">
   document.write(str);
   document.write(' <B>'+name+'</B>');
</script>
</PRE>
<HR>
#@repeatEnd#java@varNotInit;

#@repeat#java@varInit#java@varInitName;
<A NAME="#varInitDetail#@repeatIndex;"><!-- --></A><H3>
<script language="javascript">
   str = "#@java@varInitName;";
   str = str.substring(0, str.length-1);
   while (str.lastIndexOf(' ')==str.length-1) str = str.substring(0,
str.length-1);
   name = str.substring(str.lastIndexOf(' ')+1);
   str = str.substring(0, str.lastIndexOf(' '));
   document.write(name);
</script>
</H3>
<PRE>
<script language="javascript">
   document.write(str);
   document.write(' <B>'+name+'</B>');
</script>
<!--<br>
```

```
Initial value: #@java@varInit;-->
</PRE>
<HR>
#@repeatEnd#java@varInit#java@varInitName;
</span>

<!-- ========= CONSTRUCTOR DETAIL ======== -->

<A NAME="constructor_detail"><!-- --></A>
<TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
id="constructorDetail">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
<TD COLSPAN=1><FONT SIZE="+2">
<B>Constructor Detail</B></FONT></TD>
</TR>
</TABLE>

<span id="constructor_detail_text">
#@repeat#java@function;
<A NAME="consDetail#@repeatIndex;"><!-- --></A><span
id="consDetailText#@repeatIndex;"><H3>
<script language="javascript">
  str = "#@java@function;";
  inside = str.substring(str.indexOf('('));
  tmp = str.substring(0, str.indexOf('('));
  while (tmp.lastIndexOf(' ')==tmp.length-1) tmp = tmp.substring(0,
tmp.length-1);
  index = tmp.lastIndexOf(' ');
  tmp2 = str.substring(0, index);
  name = tmp.substring(index);
  pos = str.indexOf(getClassName());
  symbolStr = "\n\r\t (";
  //if ((pos<0) || ((pos>=0) && ((symbolStr.indexOf(str.charAt(pos-1))<0) ||
(symbolStr.indexOf(str.charAt(pos+getClassName().length))<0))))
  if (!isConstructor(str))
    consDetailText#@repeatIndex;.style.display = 'none';
  document.write(name);
</script>
</H3>
<PRE>
<script language="javascript">
  document.write(tmp2+'<b>'+name+"</b>"+inside);
</script></PRE>
<HR>
</span>
#@repeatEnd#java@function;
</span>

<!-- ============ METHOD DETAIL ========== -->

<A NAME="method_detail"><!-- --></A>
<TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
id="methodDetail">
<TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
<TD COLSPAN=1><FONT SIZE="+2">
<B>Method Detail</B></FONT></TD>
</TR>
</TABLE>

<span id="method_detail_text">
#@repeat#java@function;
```

```
<A NAME="funcDetail#@repeatIndex;"> </A>
<span id="funcDetailText#@repeatIndex;"> <H3>
<script language="javascript">
  str = "#@java@function;";
  inside = str.substring(str.indexOf('('));
  tmp = str.substring(0, str.indexOf('('));
  while (tmp.lastIndexOf(' ')==tmp.length-1) tmp = tmp.substring(0,
tmp.length-1);
  index = tmp.lastIndexOf(' ');
  tmp2 = str.substring(0, index);
  name = tmp.substring(index);
  pos = str.indexOf(getClassName());
  symbolStr = "\n\r\t (";
  //if ((pos>=0) && (symbolStr.indexOf(str.charAt(pos-1))>=0) &&
(symbolStr.indexOf(str.charAt(pos+getClassName().length))>=0))
  if (isConstructor(str))
     funcDetailText#@repeatIndex;.style.display = 'none';

  document.write(name);
</script>
</H3>
<PRE>
<script language="javascript">
  document.write(tmp2+'<b>'+name+"</b>"+inside);
</script></PRE>
<HR>
</span>
#@repeatEnd#java@function;
</span>

<!-- ========= END OF CLASS DATA ========= -->
```

Приложение 5. Пример шаблона документации к основному блоку файла *Java*-проекта на основе тегов

```
<inetDoc:replace property="FILE_ABSOLUTE_NAME" replacewhat="[^\\]"
replaceto="" var="file" />
<inetDoc:replaceOnce property="file" replacewhat="\\" replaceto="..\"
var="file" />
<inetDoc:replace property="file" replacewhat="\\\\" replaceto="\..\"
var="file" />
<inetDoc:exist property="package">

<!-- ======== START OF CLASS DATA ======== -->

<H2>
  <FONT SIZE="-1"><inetDoc:write property="package" /></FONT>
  <BR>
  <inetDoc:repeat><inetDoc:exist property="classtype">
    <inetDoc:replace property="classtype" replacewhat="public" replaceto=""
    var="type" />
    <inetDoc:replace property="type" replacewhat="abstract" replaceto=""
    var="type" />
    <inetDoc:replace property="type" replacewhat="[\s\t\r\n]" replaceto=""
    var="type" />
    <inetDoc:replace property="type" replacewhat="class" replaceto="Class"
    var="type" />
    <inetDoc:replace property="type" replacewhat="interface"
    replaceto="Interface" var="type" />
    <inetDoc:write property="type" />
  </inetDoc:exist></inetDoc:repeat>
  <inetDoc:repeat>
    <inetDoc:exist property="classname">
      <inetDoc:write property="classname" />
  </inetDoc:exist></inetDoc:repeat>
</H2><PRE>
  java.lang.Object
    <inetDoc:exist property="extends">
      <inetDoc:exist property="import"><inetDoc:repeat property="import"
      var="import">
        <inetDoc:replace property="import" replacewhat="\w+\." replaceto=""
        var="imp" />
        <inetDoc:match property="imp" expression="${extends}">
          <BR>   
          <img src="<inetDoc:write property="file"/>resources\inherit.gif">
          <inetDoc:write property="import" />
    </inetDoc:match></inetDoc:repeat></inetDoc:exist></inetDoc:exist>
    <BR>   
    <img src="<inetDoc:write property="file"/>resources\inherit.gif">
    <b><inetDoc:write property="package"/>.<inetDoc:repeat><inetDoc:exist
    property="classname"><inetDoc:write property="classname"/>
    </inetDoc:exist></inetDoc:repeat>
</b></PRE>
<DL></DL>
<HR><DL>
<DT><inetDoc:repeat><inetDoc:exist property="classtype">
  <inetDoc:write property="classtype" />
```

```
</inetDoc:exist><inetDoc:exist property="classname">
  <b>
    <inetDoc:write property="classname"/>
  </b></inetDoc:exist></inetDoc:repeat><DT>
<inetDoc:repeat>
  <inetDoc:exist property="Created">
    Created <i><inetDoc:write property="Created"/></i>
  </inetDoc:exist>
  <inetDoc:exist property="User">
    User: <i><inetDoc:write property="User"/></i>
  </inetDoc:exist>
  <inetDoc:exist property="Date">
    Date: <i><inetDoc:write property="Date"/></i>
  </inetDoc:exist>
  <inetDoc:exist property="Time">
    Time: <i><inetDoc:write property="Time"/></i>
</inetDoc:exist></inetDoc:repeat></DL>
<P>
  <inetDoc:repeat><inetDoc:match property="BLOC_TYPE" expression="class">
    <inetDoc:exist property="description">
      <DD><DL><inetDoc:write property="description"/>
      </DL></DD>
  </inetDoc:exist></inetDoc:match></inetDoc:repeat>
<P><HR><P>

<!-- =========== FIELD SUMMARY =========== -->

<inetDoc:repeat>
  <inetDoc:exist property="varname">
    <inetDoc:save var="varexist" expression="true" level="3"/>
</inetDoc:exist></inetDoc:repeat>
<inetDoc:exist property="varexist">
  <A NAME="field_summary"><!-- --></A>
  <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
  id="fieldSummary">
  <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
    <TD COLSPAN=2><FONT SIZE="+2">
    <B>Field Summary</B></FONT></TD>
  </TR>
  <inetDoc:repeat>
    <inetDoc:exist property="var">
      <inetDoc:repeat property="var" var="variable">
        <TR BGCOLOR="white" CLASS="TableRowColor">
        <TD ALIGN="right" VALIGN="top" WIDTH="1%"><FONT SIZE="-1">
        <CODE>
          <inetDoc:replace property="variable" replacewhat="=(.|\n)"
          replaceto="=" var="leftOfVar"/>
          <inetDoc:replace property="leftOfVar" replacewhat="=" replaceto=""
          var="leftOfVar"/>
          <inetDoc:repeat property="varname" var="vname">
            <inetDoc:match property="REPEAT_INDEX1"
            expression="${REPEAT_INDEX2}">
              <inetDoc:replace property="leftOfVar" replacewhat=";"
              replaceto="" var="leftOfVar2"/>
              <inetDoc:replace property="vname" replacewhat="\s"
              replaceto="" var="vname2"/>
              <inetDoc:replace property="leftOfVar2"
              replacewhat="${vname2}" replaceto="" var="leftOfVar2"/>
              <inetDoc:write property="leftOfVar2"/>
          </inetDoc:match></inetDoc:repeat>
        </CODE></FONT></TD>
```

```
          <TD><CODE><B>
          <A HREF="#<inetDoc:repeat property="varname" var="vname">
          <inetDoc:match property="REPEAT_INDEX1"
          expression="${REPEAT_INDEX2}">
            <inetDoc:replace property="vname" replacewhat="[\n\s\t\r]"
            replaceto="" var="vname"/>
            <inetDoc:save var="vvname" expression="${vname}" level="2"/>
            <inetDoc:write property="vname"/>
          </inetDoc:match></inetDoc:repeat>">
          <inetDoc:repeat property="varname" var="vname">
            <inetDoc:match property="REPEAT_INDEX1"
            expression="${REPEAT_INDEX2}">
              <inetDoc:write property="vname"/>
          </inetDoc:match></inetDoc:repeat>
          </A></B></CODE>
          <BR>
                     
          <inetDoc:exist property="description">
            <inetDoc:match property="BLOC_TYPE" expression="bloc">
              <inetDoc:match property="REPEAT_INDEX1" expression="0">
                <inetDoc:match property="element"
                expression="(.|\n)*${vvname}(.|\n)*">
                  <inetDoc:write property="description"/>
          </inetDoc:match></inetDoc:match></inetDoc:match></inetDoc:exist>
          </TD></TR>
  </inetDoc:repeat></inetDoc:exist></inetDoc:repeat>
  </TABLE> 
</inetDoc:exist>

<!-- ======== CONSTRUCTOR SUMMARY ======== -->

<inetDoc:repeat>
  <inetDoc:exist property="classname">
    <inetDoc:save expression="${classname}" var="classname" level="3"/>
</inetDoc:exist></inetDoc:repeat>
<inetDoc:repeat>
  <inetDoc:exist property="functionname">
    <inetDoc:save var="functionexist" expression="true" level="3"/>
</inetDoc:exist></inetDoc:repeat>
<inetDoc:exist property="functionexist">
  <A NAME="constructor_summary"><!-- --></A>
  <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
  id="constructorSummary">
  <tbody id="constructor_summary_table">
  <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
    <TD COLSPAN=2><FONT SIZE="+2">
    <B>Constructor Summary</B></FONT></TD>
  </TR>
  <inetDoc:repeat>
    <inetDoc:exist property="function"><inetDoc:repeat
  property="functionname" var="func"><inetDoc:match property="func"
  expression="${classname}">
  <TR BGCOLOR="white" CLASS="TableRowColor" id="constructor">
<TD><CODE><B><A HREF="#<inetDoc:repeat property="function" var="fname">
  <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
    <inetDoc:replace property="fname" replacewhat="[\n\s\t\r]" replaceto=""
  var="fname"/>
      <inetDoc:write property="fname"/>
</inetDoc:match></inetDoc:repeat>"><inetDoc:write
property="functionname"/></A></B>
(<inetDoc:repeat property="functioninside" var="inside">
```

```
  <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
    <inetDoc:write property="inside"/>
</inetDoc:match></inetDoc:repeat>)
</CODE><BR>
           
<inetDoc:exist property="description"><inetDoc:match property="REPEAT_INDEX1"
expression="0">
  <inetDoc:match property="BLOC_TYPE" expression="bloc">
    <inetDoc:match property="element" expression="(.|\n)*${func}(.|\n)*">
      <inetDoc:write property="description"/>
</inetDoc:match></inetDoc:match></inetDoc:match></inetDoc:exist></TD>
</TR></inetDoc:match></inetDoc:repeat></inetDoc:exist></inetDoc:repeat>
</tbody>
</TABLE>
 
</inetDoc:exist>

<!-- ========== METHOD SUMMARY =========== -->

<inetDoc:exist property="functionexist">
  <A NAME="method_summary"><!-- --></A>
  <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
  id="methodSummary"><tbody id="method_summary_table">
  <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
    <TD COLSPAN=2><FONT SIZE="+2">
    <B>Method Summary</B></FONT></TD>
  </TR>
  <inetDoc:repeat>
    <inetDoc:exist property="functionname">
    <inetDoc:repeat property="functionname" var="func">
    <inetDoc:notMatch property="func" expression="${classname}">
      <TR BGCOLOR="white" CLASS="TableRowColor">
      <TD ALIGN="right" VALIGN="top" WIDTH="1%"><FONT SIZE="-1">
      <CODE>
      <inetDoc:repeat property="function" var="fname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:replace property="fname" replacewhat="${func}" replaceto=""
        var="func"/>
        <inetDoc:replace property="func" replacewhat="\((\n|.)*\)"
        replaceto="" var="func"/>
        <inetDoc:replace property="func" replacewhat="public" replaceto=""
        var="func"/>
        <inetDoc:write property="func"/>
      </inetDoc:match></inetDoc:repeat>
      </CODE></FONT></TD>
      <TD><CODE><B><A HREF="#<inetDoc:repeat property="function"
      var="fname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:replace property="fname" replacewhat="[\n\s\t\r]"
        replaceto="" var="fname"/>
        <inetDoc:write property="fname"/>
      </inetDoc:match></inetDoc:repeat>">
      <inetDoc:write property="func"/></A></B> (
      <inetDoc:repeat property="functioninside" var="inside">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:write property="inside"/>
      </inetDoc:match></inetDoc:repeat>)</CODE>
      <BR>
                 <ine
      tDoc:exist property="description">
      <inetDoc:match property="REPEAT_INDEX1" expression="0">
```

```
        <inetDoc:match property="BLOC_TYPE" expression="bloc">
        <inetDoc:match property="element" expression="(.|\n)*${func}(.|\n)*">
          <inetDoc:write property="description"/>
        </inetDoc:match></inetDoc:match></inetDoc:match></inetDoc:exist>
        </TD>
      </TR></inetDoc:notMatch></inetDoc:repeat></inetDoc:exist>
   </inetDoc:repeat>
   </tbody>
   </TABLE>

    

</inetDoc:exist>

<P>

<a name="rule"/>

<!-- ============ FIELD DETAIL =========== -->

<inetDoc:exist property="varexist">
   <A NAME="field_detail"><!-- --></A>
   <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
   id="fieldDetail">
      <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
        <TD COLSPAN=1><FONT SIZE="+2">
        <B>Field Detail</B></FONT></TD>
      </TR>
   </TABLE>

   <inetDoc:repeat>
   <inetDoc:exist property="var">
   <inetDoc:repeat property="var" var="variable">
      <A NAME="#<inetDoc:repeat property="varname" var="vname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:replace property="vname" replacewhat="[\n\s\t\r]"
        replaceto="" var="vname"/>
        <inetDoc:write property="vname"/>
      </inetDoc:match></inetDoc:repeat>"><!-- --></A><H3>
      <inetDoc:repeat property="varname" var="vname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:save var="vvname" expression="${vname}" level="2"/>
        <inetDoc:write property="vname"/>
      </inetDoc:match></inetDoc:repeat>
      </H3>
      <PRE>
      <inetDoc:replace property="variable" replacewhat="=(.|\n)" replaceto="="
      var="leftOfVar"/>
      <inetDoc:replace property="leftOfVar" replacewhat="=" replaceto=""
      var="leftOfVar"/>
      <inetDoc:repeat property="varname" var="vname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:replace property="leftOfVar" replacewhat=";" replaceto=""
        var="leftOfVar2"/>
        <inetDoc:replace property="vname" replacewhat="\s" replaceto=""
        var="vname2"/>
        <inetDoc:replace property="leftOfVar2" replacewhat="${vname2}"
        replaceto="" var="leftOfVar2"/>
        <inetDoc:write property="leftOfVar2"/>
        <b><inetDoc:write property="vname"/></b>
      </inetDoc:match></inetDoc:repeat>
```

```
      </PRE>
      <inetDoc:match property="BLOC_TYPE" expression="bloc">
      <inetDoc:match property="REPEAT_INDEX1" expression="0">
      <inetDoc:match property="element" expression="(.|\n)*${vvname}(.|\n)*">
        <inetDoc:exist property="description">
          <DD><DL>
          <inetDoc:write property="description"/></DL></DD><P>
        </inetDoc:exist>
        <inetDoc:exist property="param">
          <DD><DL><DT><b>Parameters:</b>
          <inetDoc:repeat property="param" var="pp">
            <DD><inetDoc:write property="pp"/></DD>
          </inetDoc:repeat></DL></DD><P>
        </inetDoc:exist>
        <inetDoc:exist property="return">
          <DD><DL><DT><b>Returns:</b><DD>
          <inetDoc:write property="return"/>
          </DD></DL></DD><P>
        </inetDoc:exist>
      </inetDoc:match></inetDoc:match></inetDoc:match>
      <DL>
      <inetDoc:match property="variable"
      expression="(\n|.)*static[\n\r\t\s](\n|.)*">
        <DT><B>See Also:</B><DD>
        <inetDoc:replace property="FILE_ABSOLUTE_NAME" replacewhat="[^\\]"
        replaceto="" var="file"/>
        <inetDoc:replaceOnce property="file" replacewhat="\\" replaceto="..\"
        var="file"/>
        <inetDoc:replace property="file" replacewhat="\\\\" replaceto="\..\"
        var="file"/>
        <A HREF="<inetDoc:write property="file" />constant-values.html#
        <inetDoc:repeat>
        <inetDoc:exist property="package">
          <inetDoc:write property="package"/>
        </inetDoc:exist></inetDoc:repeat>.
        <inetDoc:exist property="thisclassname">
          <inetDoc:write property="thisclassname"/>
        </inetDoc:exist>.
        <inetDoc:repeat property="varname" var="vname">
        <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
          <inetDoc:replace property="vname" replacewhat="[\n\s\t\r]"
          replaceto="" var="vname"/>
          <inetDoc:write property="vname"/>
        </inetDoc:match></inetDoc:repeat>">
        Constant Field Values</A></DD>
      </inetDoc:match>
      </DL><HR>
    </inetDoc:repeat></inetDoc:exist></inetDoc:repeat>

</inetDoc:exist>

<!-- ========= CONSTRUCTOR DETAIL ======== -->

<inetDoc:exist property="functionexist">
  <A NAME="constructor_detail"><!-- --></A>
  <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
  id="constructorDetail">
    <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
      <TD COLSPAN=1><FONT SIZE="+2">
        <B>Constructor Detail</B></FONT></TD>
    </TR>
```

```
  </TABLE>
  <inetDoc:repeat>
  <inetDoc:exist property="functionname">
    <inetDoc:repeat property="functionname" var="func">
    <inetDoc:match property="func" expression="${classname}">
      <A NAME="#<inetDoc:repeat property="function" var="fname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
        <inetDoc:replace property="fname" replacewhat="[\n\s\t\r]"
        replaceto="" var="fname"/>
        <inetDoc:write property="fname"/>
      </inetDoc:match></inetDoc:repeat>">
      <!-- --></A><H3>
      <inetDoc:write property="func"/>
      </H3>
      <PRE>
      <inetDoc:repeat property="function" var="fname">
      <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
      <inetDoc:replace property="fname" replacewhat="${func}" replaceto=""
      var="func"/>
      <inetDoc:replace property="func" replacewhat="\((\n|.)*\)"
      replaceto="" var="func"/>
      <inetDoc:replace property="func" replacewhat="[\s\n\r\t][\s\n\r\t]"
      replaceto=" " var="func"/>
      <inetDoc:write property="func"/>
    </inetDoc:match></inetDoc:repeat>
    <b><inetDoc:write property="func"/></b>
    <inetDoc:repeat property="function" var="fname">
    <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
      <inetDoc:replace property="fname" replacewhat="(\n|.)\(" replaceto="("
      var="func"/>
      <inetDoc:write property="func"/>
    </inetDoc:match></inetDoc:repeat>
    </PRE>
    <inetDoc:match property="BLOC_TYPE" expression="bloc">
    <inetDoc:match property="REPEAT_INDEX1" expression="0">
    <inetDoc:match property="element" expression="(.|\n)*${func}(.|\n)*">
      <inetDoc:exist property="description">
        <DD><DL><inetDoc:write property="description"/></DD></DL><P>
      </inetDoc:exist>
      <inetDoc:exist property="param">
        <DD><DL><DT><b>Parameters:</b>
        <inetDoc:repeat property="param" var="pp">
          <DD><inetDoc:write property="pp"/></DD>
        </inetDoc:repeat></DL></DD><P>
      </inetDoc:exist>
      <inetDoc:exist property="return">
        <DD><DL><DT><b>Returns:</b><DD>
        <inetDoc:write property="return"/>
        </DD></DL></DD><P>
      </inetDoc:exist>
    </inetDoc:match></inetDoc:match></inetDoc:match><DL>
    <DL>
    </DL>
    </DL>
    <HR>
  </inetDoc:match></inetDoc:repeat></inetDoc:exist></inetDoc:repeat>
</inetDoc:exist>
<!-- ============ METHOD DETAIL ========== -->
<inetDoc:exist property="functionexist">
  <A NAME="method_detail"><!-- --></A>
```

```
  <TABLE BORDER="1" WIDTH="100%" CELLPADDING="3" CELLSPACING="0" SUMMARY=""
  id="methodDetail">
    <TR BGCOLOR="#CCCCFF" CLASS="TableHeadingColor">
      <TD COLSPAN=1><FONT SIZE="+2">
        <B>Method Detail</B></FONT></TD>
    </TR>
  </TABLE>
  <inetDoc:repeat>
  <inetDoc:exist property="functionname">
  <inetDoc:repeat property="functionname" var="func">
  <inetDoc:notMatch property="func" expression="${classname}">
    <A NAME="#<inetDoc:repeat property="function" var="fname">
    <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
    <inetDoc:replace property="fname" replacewhat="[\n\s\t\r]" replaceto=""
    var="fname"/>
      <inetDoc:write property="fname"/>
    </inetDoc:match></inetDoc:repeat>">
    <!-- --></A><H3>
    <inetDoc:write property="func"/>
    </H3>
    <PRE>
    <inetDoc:repeat property="function" var="fname">
    <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
      <inetDoc:replace property="fname" replacewhat="${func}" replaceto=""
      var="func"/>
      <inetDoc:replace property="func" replacewhat="\((\n|.)*\)"
      replaceto="" var="func"/>
      <inetDoc:replace property="func" replacewhat="[\s\n\r\t][\s\n\r\t]"
      replaceto=" " var="func"/>
      <inetDoc:write property="func"/>
    </inetDoc:match></inetDoc:repeat>
    <b><inetDoc:write property="func"/></b>
    <inetDoc:repeat property="function" var="fname">
    <inetDoc:match property="REPEAT_INDEX1" expression="${REPEAT_INDEX2}">
      <inetDoc:replace property="fname" replacewhat="(\n|.)\(" replaceto="("
      var="func"/>
      <inetDoc:write property="func"/>
    </inetDoc:match></inetDoc:repeat>
    </PRE>
    <inetDoc:match property="BLOC_TYPE" expression="bloc">
    <inetDoc:match property="REPEAT_INDEX1" expression="0">
    <inetDoc:match property="element" expression="(.|\n)*${func}(.|\n)*">
      <inetDoc:exist property="description">
        <DD><DL><inetDoc:write property="description"/></DD></DL><P>
      </inetDoc:exist>
      <inetDoc:exist property="param">
        <DD><DL><DT><b>Parameters:</b>
        <inetDoc:repeat property="param" var="pp">
          <DD><inetDoc:write property="pp"/></DD>
        </inetDoc:repeat></DL></DD><P>
      </inetDoc:exist>
      <inetDoc:exist property="return">
        <DD><DL><DT><b>Returns:</b><DD>
        <inetDoc:write property="return"/></DD></DL></DD><P>
      </inetDoc:exist>
    </inetDoc:match></inetDoc:match></inetDoc:match>
    <DL><DL></DL></DL><HR>
  </inetDoc:notMatch></inetDoc:repeat></inetDoc:exist></inetDoc:repeat>
</inetDoc:exist>
<!-- ========= END OF CLASS DATA ========= -->
</inetDoc:exist>
```